USERS

★★★★★ DESARROLLOS REALES PARA LA VERSIÓN 2010

VISUAL BASIC

GUÍA DEFINITIVA DEL PROGRAMADOR

VISUAL STUDIO 2010 Y .NET FRAMEWORK 4

APLICACIONES WEB CON ASP.NET

OPTIMIZAR EL USO DE LOS RECURSOS DEL S.O.

DIFERENCIAS Y SIMILITUDES EN LA SINTAXIS CON C#

BASES DE DATOS SQL SERVER 2008



por Fernando Omar Luna

USERS MANUALES USERS MANUALES USERS MANUALES USERS MANUALES USERS

LAS MEJORES PRÁCTICAS PARA EL ÉXITO PROFESIONAL

VISUAL BASIC

GUÍA DEFINITIVA DEL PROGRAMADOR

por Fernando O. Luna

RedUSERS



TÍTULO: Visual Basic

AUTOR: Fernando O. Luna

COLECCIÓN: Manuales Users

FORMATO: 17 x 24 cm

PÁGINAS: 352

ISBN 978-987-1773-57-2

Luna, Fernando O.
Visual Basic. - 1a ed. - Buenos Aires : Fox Andina; Dalaga, 2011.
v. 215, 352 p. ; 24x17 cm. - (Manual users)

ISBN 978-987-1773-57-2

1. Informática. I. Título
CDD 005.3

Fernando O. Luna



Fernando Omar Luna es Analista programador universitario. En 1994 tuvo contacto con las computadoras por primera vez, y luego de haber probado todo, llegó el turno de la programación; entonces descubrió que su escaso gusto por las matemáticas siendo niño se había revertido por completo gracias al inquietante mundo de los algoritmos. Antes de ingresar en el ámbito de la computación, Fernando se graduó como Operador auxiliar de planta transmisora de radio, pasión que devino de sus casi siete años como DJ. Desde 1998 trabaja como programador en varias compañías (industrias nacionales y empresas de salud). Actualmente reparte su tiempo entre el desarrollo de sistemas de gestión y su pasión por la escritura, que lo llevó a colaborar con la revista Power Users y algunos blogs españoles, además de tener su propio portal tecnológico: f-digital.
Blog: **http://blog.f-digital.com.ar**
E-mail: **fernando@f-digital.com.ar**

Dedicatoria

Este libro fue escrito para mis hijos, quienes me brindan amor en cada momento que disfrutamos juntos de la vida.

Agradecimientos

A quienes me dan amor incondicional: a Nico y July, que son el combustible que me mantiene en marcha; a mi familia, que respeta mis tiempos y obligaciones; a Julio, que hizo el esfuerzo enorme de traer a mi habitación un clon 286 usado con 2 MB de RAM, y que, a su vez, junto a Nélida, desde algún rincón del éter, siguen iluminando mi camino por esta vida.
A la editorial, a Nicolás Kestelboim y a Mariel Cerra, quienes se armaron de paciencia y buena onda para que este proyecto llegara a transformarse en un libro.

PRÓLOGO

Llegando al primer lustro de la década de los noventa, las computadoras comenzaron a llamarme la atención. Luego de haber hecho un curso de reparación de PC, posterior a dos años de introducción a la electrónica y la especialización en radio y audio, aquellos gabinetes metálicos y por ese entonces, pesados, aún contenían disqueteras de 5 ¼, las cuales fueron poco a poco reemplazadas por las de 3 ½. El reciente Windows 3.11 para trabajo en grupos era el sistema operativo de moda, y solo aquellos que eran inquietos como yo sabían que eran contados los lenguajes de programación que había en el mercado y que podían lograr crear un software en muy poco tiempo. En la era donde programar era una verdadera profesión y solo unos pocos tenían el don de armar un sistema de gestión "agiornado" con una interfaz relativamente clara, hizo su aparición Visual Basic. De la mano de Microsoft, este lenguaje de programación e IDE "todo en uno" se tomaba el desafío de captar el interés de aquellos programadores que le daban cientos de horas de sus vidas a Clipper, Dbase y Paradox. Pocos eran los valientes que se les animaban a estos lenguajes y sin embargo un escaso grupo reducido de visionarios sabían que el mundo informático tarde o temprano dejaría de lado la neutral pantalla negra y blanca para darle paso a la nueva era, la era Visual.
El mundo tecnológico cambió gracias a esos hombres que decidieron contra viento y marea elegir la innovación y apostar por un nuevo futuro, el futuro gráfico, el futuro Visual. Gracias a la era iniciada por Visual Basic, se creó un significativo punto de inflexión entre los desarrollos a pulmón y los desarrollos inteligentes. Además las cientos de líneas de código necesarias para la creación de un menú o de un botón lineal se transformaron rápidamente en algunos clics y combinaciones de teclas, demostrando así que la productividad y la inteligencia van de la mano y no por veredas paralelas.

Fernando Luna

EL LIBRO DE UN VISTAZO

Este libro está enfocado a usuarios principiantes e intermedios que deseen conocer las herramientas de Microsoft con el objetivo desarrollar software para Windows y otras tecnologías móviles, a través del lenguaje ya consagrado y conocido por todos, como es Visual Basic en su versión 2010.

Capítulo 1

INTRODUCCIÓN A LA PLATAFORMA

En este capítulo, realizamos una introducción al lenguaje Visual Basic. Conoceremos su historia, sus versiones y cambios a través del tiempo. Estudiaremos con detenimiento su IDE para luego poder manejarnos cómodamente con él.

Capítulo 2

FUNDAMENTOS DE VISUAL BASIC

Comenzaremos a conocer en detalle su lenguaje, sintaxis, los tipos de datos, las palabras reservadas y la estructura que permite crear una aplicación sencilla y estéticamente agradable en pocos minutos. También comprenderemos las diferencias esenciales entre los procedimientos y las funciones del lenguaje.

Capítulo 3

MY NAMESPACE

Los espacios de nombre, conocidos como Namespaces, son lo que le da vida a toda aplicación llevada a cabo con la tecnología .NET. En este capítulo conoceremos el porqué y cuáles son los Namespaces necesarios para utilizar, dependiendo del objetivo que deseamos alcanzar. También navegaremos entre el uso de los controles visuales, que nos permiten ahorrar muchas horas de desarrollo de la interfaz GUI con tan sólo arrastrar, soltar y configurar.

Capítulo 4

MANEJO DE ARCHIVOS

Este capítulo nos pone de lleno el conocimiento de nuestro sistema operativo, su sistema de archivos y la organización de éste mediante directorios. Aprenderemos cómo manipular todo tipo de archivos desde el código de nuestra aplicación y también desarrollaremos un programa que nos permitirá crear, modificar y guardar archivos de texto enriquecido.

Capítulo 5

BASES DE DATOS

Las bases de datos son esenciales en el mundo del software. Cualquier aplicación, sea cual fuere su objetivo, depende de una base de datos que le permita almacenar y consultar información de parámetros, entre otras cosas. También los sistemas de gestión hacen un uso exhaustivo de estas bases de datos para manipular información que almacenan los usuarios.

Capítulo 6

DEPURACIÓN Y MANEJO DE ERRORES

Todo software que lleva aunque sea diez líneas de código no está exento de contener errores en su programación. Este capítulo busca ayudar al usuario a prevenir al máximo la aparición de errores en el software y de ocurrir esto, saber cómo controlarlos sin que el software pierda el control total de ejecución.

Capítulo 7

APLICACIONES ASP.NET

La llegada de Internet a nuestras vidas hizo que el mundo de la programación dejara de limitarse a un simple escritorio y sistema operativo. Este capítulo le brinda al usuario la información necesaria para adentrarse en el mundo de desarrollo de software que corra sobre Internet o sobre los navegadores web.

Capítulo 8

XAML Y WPF

El mundo web no trajo solo consigo el desarrollo web de aplicaciones que corran sobre la gran red de redes. También trajo de su mano nuevas tecnologías que buscan reemplazar los clásicos desarrollos de escritorio. En este capítulo conoceremos a XAML y WPF, dos tecnologías que llegaron para quedarse en nuestras computadoras.

Apéndice A

Así como la teoría es fundamental para la comprensión de los conceptos básicos de todo objetivo en la vida. La práctica también requiere su lugar. En este apéndice, conoceremos la manera de realizar aplicaciones sencillas y efectivas con algunos pocos clics del mouse y líneas de código. Verdaderas aplicaciones hechas en tiempo récord.

Apéndice B

Windows Phone 7 es la nueva apuesta de Microsoft para el mundo móvil, y Visual Studio 2010 busca acompañar los desarrollos de software para este novedoso y reestructurado sistema operativo para celulares. Este apéndice nos permitirá conocer los elementos necesarios y de qué manera podemos desarrollar aplicaciones para el sistema operativo Móvil de Microsoft.

Apéndice C

C# se está poniendo de moda. En los últimos cinco años ha crecido de manera exorbitante el uso de este lenguaje de programación. En el último apéndice del libro, sabremos cuáles son las principales diferencias y similitudes entre Visual Basic y Visual C#.

Servicios al lector

En el último apartado deeste libro, tenemos a nuestra disposición el índice temático, donde encontraremos, de manera rápida y eféctiva, los principales conceptos de la obra.



INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual encontrará una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados.
Cada recuadro está identificado con uno de los siguientes iconos:

 CURIOSIDADES E IDEAS

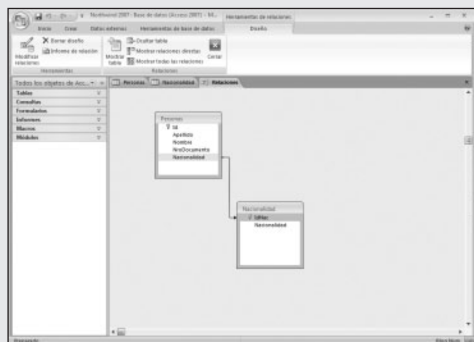
 ATENCIÓN

 DATOS ÚTILES Y NOVEDADES

 SITIOS WEB

CONTENIDO

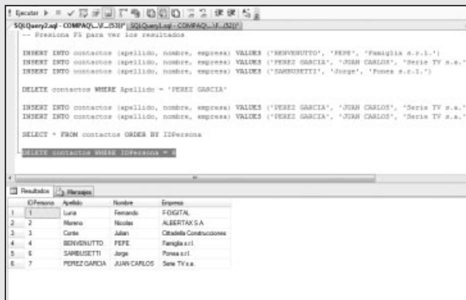

Form1
Button1
El nombre de columna 'Telefono' no es válido.
Aceptar

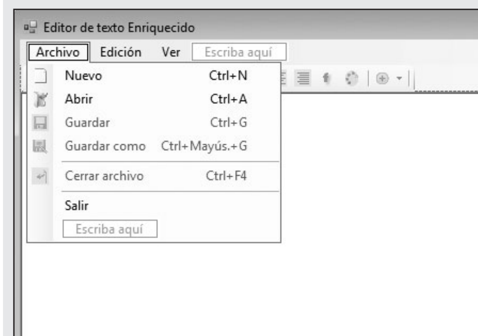


Capítulo 4

Capítulo 5

Capítulo 6




Editor de texto Enriquecido
Archivo
Edición
Ver
Escriba aquí
Nuevo Ctrl+N
Abrir Ctrl+A
Guardar Ctrl+G
Guardar como Ctrl+Mayús.+G
Cerrar archivo Ctrl+F4
Salir
Escriba aquí

INTRODUCCIÓN

El mundo informático es una pasión que cada día atrapa a más personas, quienes usan programas destinados a cubrir diversas necesidades.
Si bien el software no se palpa, es real; y por más sencillo que sea su cometido, requirió tiempo, ingeniería e idealización de una o más personas para implementarlo.
Año a año se incrementa la demanda de expertos en el mundo binario y son contadas las empresas que no dependen de una computadora o de un sistema a medida. También el desarrollo de software está disfrutando de un crecimiento ininterrumpido en cualquier país del mundo.
Convertirse en desarrollador requiere de algunos conocimientos puntuales y de mentes bien abiertas, pero gracias a la facilidad de las herramientas de que disponemos en la era 2.0 de la computación, más la biblioteca universal de información abierta las 24 horas del día, como lo es Internet, el aprendizaje de esta profesión puede llevarse a cabo en tiempo récord.
Este libro busca orientar a todas aquellas personas que desean conocer las herramientas existentes en el mercado para desarrollar aplicaciones para Windows, Internet y las actuales variantes, como la implementación de aplicaciones para dispositivos móviles, todo basado en herramientas y tecnologías de Microsoft.
Los capítulos de esta obra se organizaron de tal manera que el lector comience conociendo desde cero el IDE de desarrollo, el lenguaje, su estructura y las herramientas adicionales para potenciarlo, en desarrollos tanto para Windows como para la Web. También se verán opciones alternativas al lenguaje BASIC, como el moderno y cada vez más aceptado C#, y gracias a él, se analizará la posibilidad de programar aplicaciones destinadas al sistema operativo para celulares Windows Phone 7.
El libro busca ser una herramienta que introduzca al lector en el mundo de la programación con el lenguaje Visual Basic, a la vez que pretende ser el material de consulta futuro para llevar a la realidad determinadas soluciones de software en un corto tiempo.

Capítulo 1

Introducción a la plataforma

En el primer capítulo de Visual Basic, repasaremos la historia de este lenguaje que fue, en muchos casos, el puntapié que inició en la programación a la mayoría de los que se dedican hoy al desarrollo de software de manera particular y profesional. Veremos también los cambios incluidos en esta versión respecto a sus antecesoras más recientes, y haremos un repaso de su entorno de trabajo para familiarizarnos con la plataforma antes de iniciar los ejercicios prácticos.

BIENVENIDOS A VISUAL BASIC 2010

En poco más de 30 años, el paradigma de la programación ha dado muchas vueltas de tuerca a beneficio de las personas, al dejar de ser un mundo estrecho y cerrado a unos pocos ingenieros, para convertirse en la pasión de millones de personas interesadas en este campo. **Microsoft**, la empresa desarrolladora de software que contribuyó mucho a cambiar el panorama de la informática, ha jugado un papel importante en este terreno, por tener una amplia visión de futuro y prever las necesidades del usuario final, quien no solo se interesa por la computación, sino que también quiere avanzar en el fascinante mundo de la programación de aplicaciones para computadoras.

Visual Basic 2010 permite llegar, de manera fácil y práctica, al desarrollo de aplicaciones de escritorio, web y otros campos que hoy nos son cada vez más cotidianos, como la telefonía celular, poniendo a disposición de los interesados la potencia y la flexibilidad de una herramienta que ya tiene más de cuarenta años.

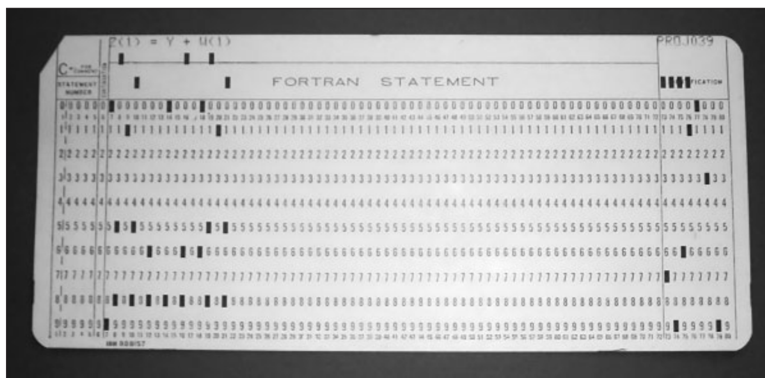


***Figura 1.** Programación de una sentencia sobre una **tarjeta perforada**, propia del lenguaje **Fortran**, en la década de 1970.*

Reseña del lenguaje

En 1964, **John Kemeny** y **Thomas Kurtz** se propusieron crear una variante del lenguaje de programación denominado **BASIC** (*Beginners All-Purpose Symbolic Instruction Code*, **Código de instrucciones simbólicas para principiantes orientado a todo propósito**). Si bien ya había en el mercado varias versiones de BASIC, la generada en **Dartmouth College** por Kemeny y Kurtz fue la que más se popularizó. BASIC llegó para reducir de una manera notable los tiempos de aprendizaje y de escritura de un programa para computadoras. Así, gracias a la visión de futuro de sus propulsores, muchos estudiantes pudieron desarrollar aplicaciones en tiempo récord, algo muy valioso para esa época.

Al momento de diseñar el lenguaje, sus creadores tomaron como filosofía ocho principios que debían destacar a BASIC por sobre los demás:

1. Ser fácil de usar.
2. Ser un lenguaje de propósito general.
3. Permitir la incorporación de características avanzadas por expertos, priorizando su facilidad para principiantes.
4. Gozar de interactividad.
5. Ofrecer claros mensajes de error.
6. Brindar rápida respuesta en programas pequeños.
7. No requerir que los usuarios tengan conocimientos sobre hardware.
8. Alejar al usuario de la complejidad del sistema operativo.

Hacia 1975, la empresa fundada por **Bill Gates** y **Paul Allen** lanzó su propia versión de BASIC, inspirada en una de las alternativas del lenguaje, creada por **Alan Cooper**, denominada **Altair BASIC**. El mercado siguió inundándose con más versiones opcionales, y para fines de la década del 70, apareció la primera adaptada a la plataforma Apple II. En 1979, Microsoft negoció vender su licencia de BASIC a varias empresas que comercializaban microcomputadoras, incluyendo a IBM, creadora en ese entonces de la computadora personal. BASIC se incorporó en los chips **ROM** de las **IBM PC**, con lo cual se puso una versión innovadora en equipos que no contaban con disco rígido, pero que sí disponían ya de una unidad de disquete.

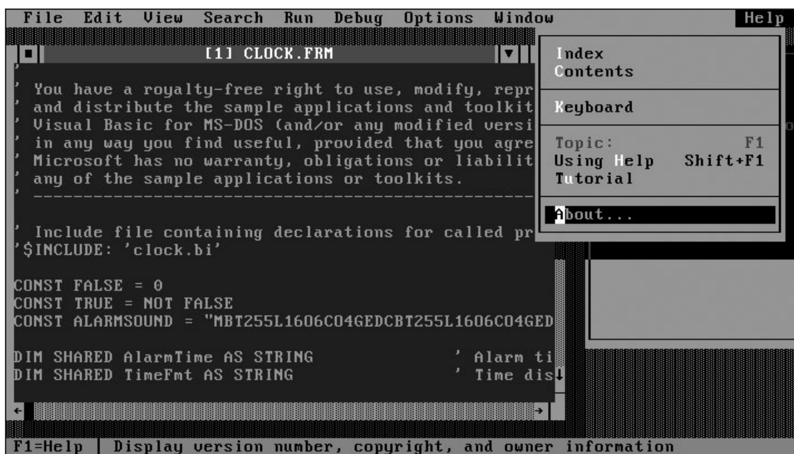


***Figura 2**. Visual Basic 1.0 fue el puente entre la versión BASIC para DOS y la versión Visual Basic para Windows.*

Años más tarde, Microsoft siguió distribuyendo una versión reducida de BASIC junto a su popular sistema operativo **MS DOS**, la cual permitía a los programadores diseñar aplicaciones que sólo podían ejecutarse a través del entorno de desarrollo utilizando una serie de instrucciones **BATCH**, para que el programa se ejecutara casi sin intervención de los usuarios poco expertos. En la entrada era donde DOS pasaba a un segundo plano. Luego la firma lanzó al mercado **Visual**

Basic 1.0, un entorno de desarrollo que facilitaba la creación de aplicaciones con menús, ventanas y botones, pero que aún corría bajo DOS. Recién en la versión 2.0, desarrollada para **Windows 3.0/3.1**, Microsoft dejó de lado el entorno gráfico construido mediante caracteres **ASCII**, para dar inicio a una era distinta: la era **RAD** de desarrollo de aplicaciones para Windows.

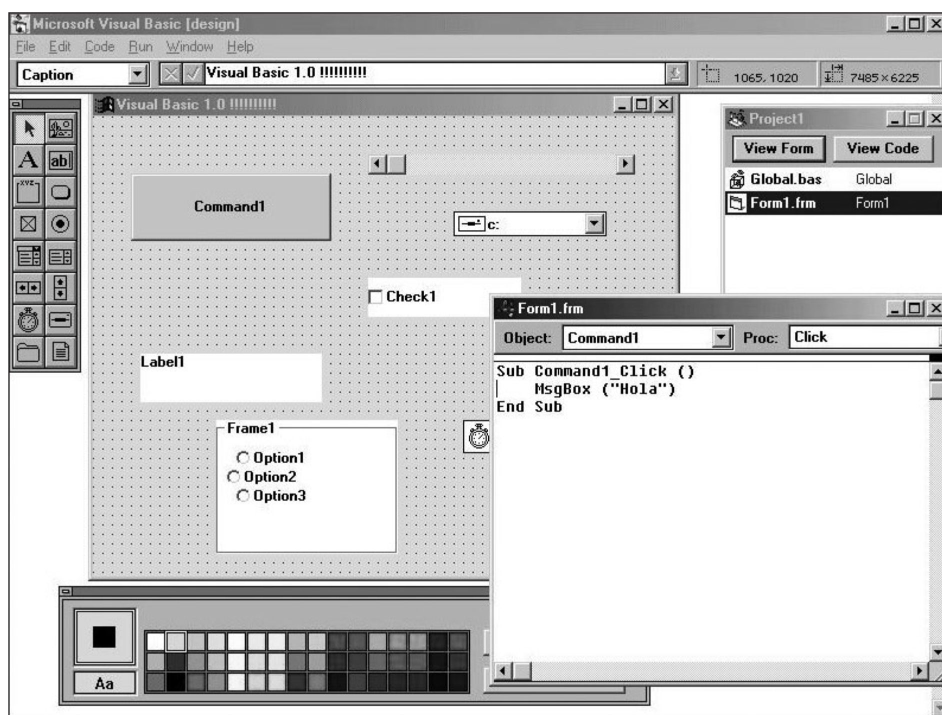


Figura 3. *Visual Basic 1.0 para Windows 2.x y 3.0 comenzó a traer el estilo de entorno de desarrollo RAD que hoy nos es familiar en el resto de los lenguajes de programación para plataformas Windows.*

Desde **Visual Basic 3.0**, Microsoft fortaleció el desarrollo de aplicaciones RAD orientado a bases de datos y dio un gran soporte al lenguaje para conectarse a cualquier base entre las más populares del mercado **(Dbase, Paradox, Fox Pro**),

{ } LIBRERÍAS VBX

A partir de la versión 2 de Visual Basic se incluyó una serie de librerías llamadas VBX (Visual Basic eXtensions), gracias a las cuales el programador podía diseñar las ventanas de la aplicación de manera rápida. De allí proviene la denominación **RAD** (*Rapid Application Development*, **desarrollo rápido de aplicaciones**) para el entorno Visual Basic.

al utilizar las librerías de enlace dinámico a través de **ODBC** (*Open Data Base Connectivity*). Para las bases de datos que no eran tan difundidas en ese momento, solo restaba que la empresa que las soportaba creara una librería DLL para que Visual Basic pudiera conectarse e interactuar con ellas y, así, leer, mostrar y escribir información en sus archivos.
La versión 4.0 llegó al mercado casi al mismo tiempo que **Windows 95**, con lo cual se lanzó una edición doble, para 16 y 32 bits, que podía instalarse en **Windows 3.1x** o **Windows 95**, con la diferencia de que todos los proyectos creados en 16 bits podían ser portados a 32 bits, pero no a la inversa.
Visual Basic 5.0 contó con una versión lite denominada **CCE** (*Control Creation Edition*), en la que no solo era posible crear librerías **DLL** y archivos ejecutables, sino que también se habilitaba a los programadores a generar controles personalizados, combinando dos o más controles **ActiveX** existentes. También se facilitaba la incorporación de nuevos eventos y propiedades a los controles predefinidos que se incluían con el entorno de desarrollo.

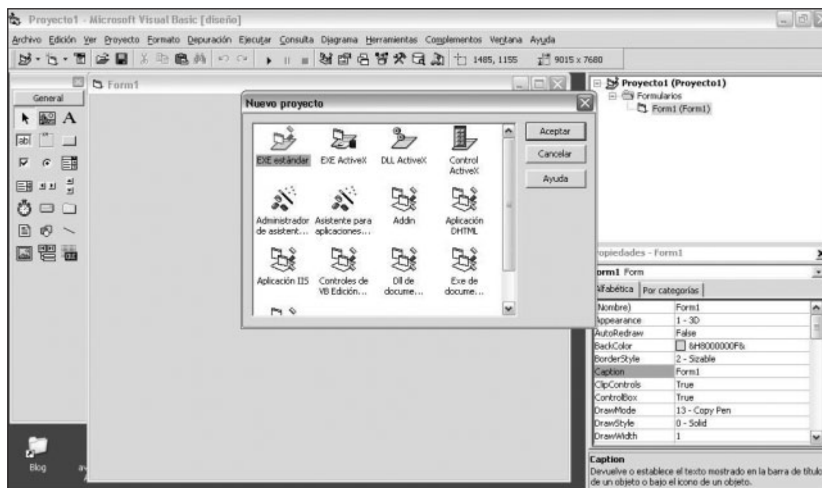


Figura 4. *Visual Basic 6.0, a más de una década de su creación, sigue gozando de popularidad en varias empresas, como bancos, aseguradoras y compañías de tarjetas de crédito.*

ACTIVEX CONTROLS

Desde la versión 4.0 de 32 bits, se dejó de dar soporte para los controles **VBX**, y se inició la era de los **OCX** (conocidos por todos como **ActiveX Controls**), junto con la capacidad de generar librerías de enlace dinámico (**.DLL**) y crear programas ejecutables (**.EXE**).

En 1998 Microsoft introdujo **Visual Basic 6.0**. En la era donde Internet se devoraba cualquier mercado, esta versión tuvo que aportar flexibilidad para dar paso al desarrollo de aplicaciones web y no solo de escritorio. Con Visual Basic 6 se podían crear controles personalizados, programas ejecutables bajo la plataforma Windows, librerías DLL y aplicaciones web, incluyendo soporte para el lenguaje ASP (*Active Server Pages*), que permitía generar páginas web dinámicas que se compilaban al momento de ser solicitadas en el servidor.
El nuevo milenio trajo consigo un giro total de sus lenguajes de programación, que se orientaron al framework .NET. Los **ActiveX** pasaron a segundo plano, y con esto, también varios problemas de seguridad que sufría Microsoft Windows.

Herramientas complementarias

Crystal Reports fue el primer software para la generación de reportes visualmente atractivos, que pertenecía a una empresa ajena a los intereses de Microsoft, pero que supo complementarse bien con este lenguaje para triunfar.
Microsoft Access, base de datos que pertenece aún hoy al paquete ofimático Microsoft Office, es otro complemento ideal para quienes buscan crear aplicaciones pequeñas que no requieran de un potente y costoso motor de base de datos.
Microsoft SQL Server, desde la versión 4.x, comenzó a ser un aliado para Visual Basic en el desarrollo de aplicaciones empresariales que requirieran un robusto motor de base de datos y necesitaran manejar grandes volúmenes de información.

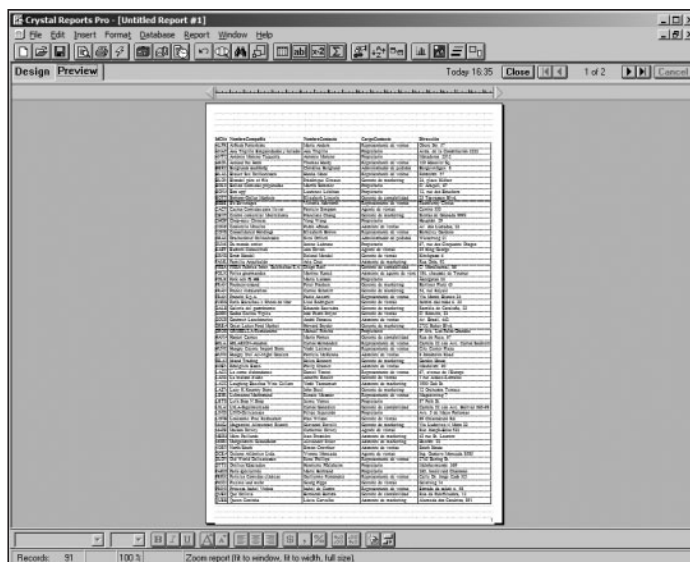


Figura 5. *Una versión **lite** de **Crystal Reports** fue distribuida con Visual Basic desde la versión 4.0 hasta la 6.0. En tan solo siete pasos visuales, era posible obtener un informe estéticamente profesional, sin ningún esfuerzo.*

Componentes de la plataforma

Microsoft Visual Studio es una plataforma para desarrollo de software integrada por varios lenguajes de programación: **Visual C++**, **Visual C#**, **ASP.NET** y **Visual Basic .NET**. Los programadores de aplicaciones independientes suelen utilizar un lenguaje específico, mientras que las empresas que se dedican a brindar soluciones de software de mediana a gran escala en general emplean más de uno, dependiendo de la necesidad del cliente. Visual Basic se integró a esta suite de lenguajes desde su versión 6.0. Microsoft pensó en desarrollar el framework .NET a fines del año 2000, y fue entonces cuando cambió la orientación de programación de todos sus lenguajes, incluyendo Visual Basic, que dejó de estar orientado a eventos y pasó a ser un lenguaje de programación orientado a objetos.

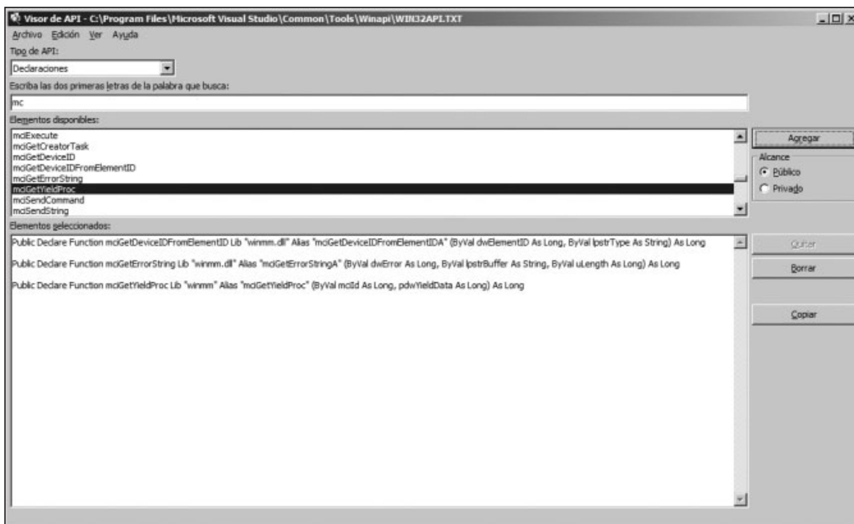


***Figura 6.** Las Windows API ayudaban a muchos lenguajes de programación a resolver la comunicación con recursos de Windows usando pocas líneas de código. Visual Basic, antes de la era .NET, incluía un Visor de API, para conseguir fácilmente el código que necesitáramos.*

La primera versión de Visual Basic .NET se lanzó en 2002, para trabajar con el framework .NET versión 1. Un año más tarde, llegó la versión 1.1 del framework,



CRYSTAL REPORTS

Es considerado como el mejor complemento para cualquier lenguaje de programación que necesita distribuir informes listos en tiempo reécord para imprimir siguiendo ciertos parámetros de una base de datos. Aunque ya no se distribuye su versión **lite** con Visual Studio, es conveniente adquirirlo para agilizar los tiempos de desarrollo.

que trajo a la luz la versión 2003 de Visual Studio. Se incluyó en ella el soporte para desarrollo de aplicaciones para dispositivos móviles, como **Windows CE** o las primeras versiones de **Windows Mobile** a través del **Compact Framework**.



Figura 7. *Compact Framework .NET, junto con los avances de la telefonía celular, permitió desarrollar aplicaciones para teléfonos móviles tan fácilmente como si se tratase de una computadora.*

El cambio sustancial que trajeron las herramientas de Visual Studio 2005 incluyó un nuevo modo de conexión a bases de datos, dado que, de la facilidad que tenía Visual Basic 6.0 para interactuar con ellas en su modelo conocido como **ADO**, Microsoft saltó a una gran complejidad que muchos programadores nunca lograron adoptar. Recién la versión 2005 de Visual Studio volvió a ofrecer un modelo de conexión a bases de datos práctico y fácil de entender. También comenzó a brindarse soporte para aplicaciones de 64 bits, mucho mejores para sacar provecho de los nuevos procesadores multinúcleo que aparecieron en el mercado.

Visual Studio 2008 añadió soporte para interactuar con el framework .NET **2.0**, **3.0** y **3.5**, e incluso mejoró las ventajas de trabajo para el nuevo sistema operativo de Microsoft, **Windows Vista**, al permitir la creación de programas del tipo **Windows Communication Foundation** (**WCF**) y **Windows Presentation Foundation** (**WPF**). WCF permite crear aplicaciones del tipo de **servicios Windows**, mientras que WPF apunta a generar interfaces de usuario más dinámicas que las permitidas por Visual Basic en ese momento.

La última versión de **Visual Studio** es la **2010**, lanzada en abril y preparada para interactuar con el framework .NET 4.0. Todas las herramientas de programación que componen este paquete de productos, incluyendo Visual Basic, fueron optimizadas para generar programas para Windows, servicios de Windows, web, móviles y para el shell de Windows: **Windows Deskbar**.



Figura 8. *Los gadgets de escritorio para Windows Vista y 7 también podrán ser desarrollados con Visual Basic 2010.*

El framework .NET

El framework .NET fue creado por Microsoft en respuesta al crecimiento ininterrumpido de los negocios en entornos web. Inspiró su filosofía en la plataforma **Java** de **Sun Microsystems** y de **PHP**, otro lenguaje de programación de páginas web que venía causando furor entre los desarrolladores de sitios dinámicos. Su estructura fue pensada para ofrecer el desarrollo práctico, seguro y robusto de aplicaciones web y Windows Forms. A la vez, permitió a la empresa darle un nuevo rumbo a su mercado de negocios, apuntando todos sus productos, hasta el sistema operativo, a centralizarse en una solución encapsulada mucho más ágil a la que proponía la estructura de objetos **COM**.

El framework .NET se convirtió en un componente de software que se puede añadir dentro del sistema operativo Windows. En él se agrupa un conjunto de soluciones predefinidas como clases que se comunican con todas las funciones del sistema operativo y, así, permite cubrir las necesidades generales para el desarrollo de aplicaciones. Se encarga de administrar, como una capa intermedia, las funciones de los programas que ejecutan diversas acciones sobre el entorno Windows.



MICROSOFT MSDN

El sitio web MSDN, http://msdn.microsoft.com, es un gran apoyo complementario para cualquier herramienta de programación. En él se incluyen muchos ejemplos con código fuente para descargar y testear en forma gratuita. Es recomendable navegar su versión en inglés, que siempre es más completa en cuanto a recursos que la ofrecida en español.

El framework se divide en dos versiones: la que se instala para interactuar dentro de la familia de sistemas operativos Windows (desde 98 en adelante), y la reducida, para funcionar en teléfonos celulares y dispositivos ultraportables, conocida como Compact Framework.

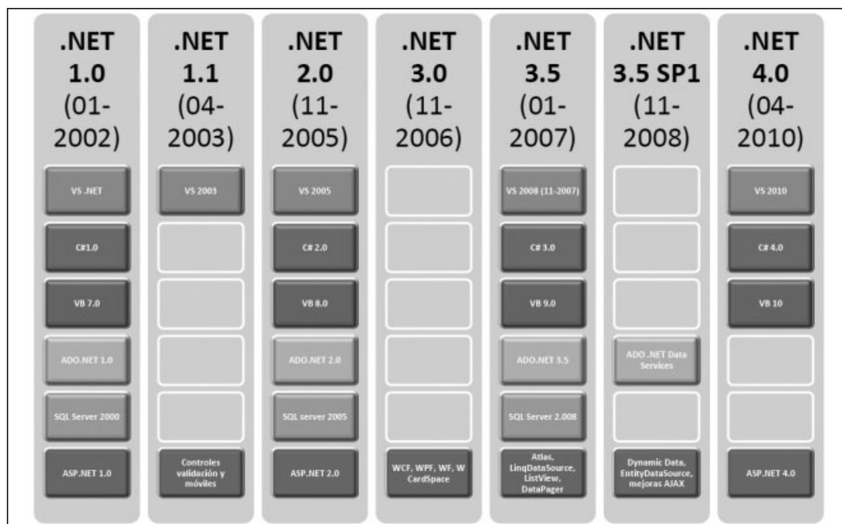


Figura 9. *Aquí podemos ver las distintas versiones del framework .NET junto con sus fechas de lanzamiento y las mejoras incluidas en cada una de ellas.*

El framework está integrado por tres componentes clave:

- La biblioteca de clases base (BCL, *Base Class Library*).
- El entorno común de ejecución de lenguajes (CLR, *Common Languages Runtime*).
- Los lenguajes de programación (principalmente, los incluidos en Visual Studio).

BCL

La biblioteca de clases base se encarga de agrupar casi todas las operaciones utilizadas en el desarrollo de una aplicación, como la interactividad con hardware, la administración de memoria, el manejo de datos (conocido como **ADO.NET**), la



WEB OFICIAL DE VISUAL STUDIO 2010

Recordemos visitar el sitio web oficial del conjunto de herramientas Visual Studio, porque en ella nos enteraremos de las nuevas versiones y fixes, comprenderemos mejor las diferencias que surgen entre sus distintas versiones, y estaremos al tanto de las futuras ediciones: **www.microsoft.com/visualstudio/en-us**.

interacción en las comunicaciones mediante el protocolo **TCP/IP** o a través de **XML**, los componentes que corren tanto en un servidor web como en la computadora cliente, la administración de los datos en memoria, las ventanas creadas por la aplicación, el manejo de excepciones (errores), los gráficos **GDI+**, la interacción entre aplicaciones, las operaciones matemáticas, los archivos de imágenes, el manejo de culturas e idiomas, la interacción con la **API** de Windows, y otras funciones más del sistema operativo.
Todo esto está estructurado dentro de **BCL** mediante espacios de nombres jerárquicos, que veremos más adelante. La biblioteca de clases base se organiza en cuatro grupos clave detallados a continuación:

- ASP.NET y servicios XML
- Windows Forms
- ADO.NET
- El entorno .NET en sí

CLR

Common Language Runtime es el núcleo del framework .NET. A través de este entorno de ejecución, las aplicaciones desarrolladas con esta tecnología son interpretadas y ejecutadas. **CLR** interpreta el código de cualquiera de los lenguajes de programación escritos para trabajar con .NET. El código de cada aplicación es compilado a través de **MSIL** (*Microsoft Intermediate Language*), un lenguaje intermedio similar al BYTECODE empleado por Java. Esta compilación se genera utilizando las especificaciones basadas en **CLS** (*Common Language Specification*), para terminar siendo ejecutado a través del compilador **JIT** (*Just In Time*), que produce el código máquina para que el programa sea interpretado por el procesador de la computadora donde corre. Así es como .NET se independiza de la plataforma en la que se ejecuta, sin importarle el tipo de hardware usado. Una vez compilada la aplicación, esta es almacenada en la memoria caché de la máquina que la ejecutó, y solo vuelve a ser recompilada en caso de que sea modificado parte de su código fuente.

Lenguajes de programación

Actualmente, el framework .NET soporta los lenguajes de programación **C#**, **Visual Basic, Object Pascal (Delphi)**, **C++**, **J#** (que se ha discontinuado a partir de la versión 2010 de Visual Studio), **Perl**, **Python**, **Fortran**, **Cobol**, **PowerBuilder**, **Prolog**, y el recientemente incorporado **F#**, entre otros. Cualquier persona o empresa puede publicar un lenguaje de programación orientado hacia la plataforma .NET siempre y cuando respete la convención propuesta por Microsoft, la cual garantiza el correcto funcionamiento de una solución basada en software dentro de la computadora en que es ejecutada.



Figura 10. *En el sitio web de Microsoft .NET Framework podemos estar al tanto de las últimas novedades, y de las nuevas descargas y parches disponibles:* **http://msdn.microsoft.com/es-ar/netframework/default.aspx**.

Diferencias entre versiones

Visual Basic 2010, como también el resto de los lenguajes que componen la suite Visual Studio 2010, dispone de diferentes versiones en el mercado: **Express Edition**, **Standard Edition**, **Professional Edition** y **Team System Edition**.

Edición Express

La edición **Express** de Visual Studio 2010 es una edición gratuita del conjunto de lenguajes orientada hacia el ámbito educativo. Con este entorno, podremos crear aplicaciones del tipo **Windows Forms**, **Windows Presentation Foundation**, **aplicaciones de consola, biblioteca de clases** y **aplicaciones del explorador de Internet WPF**.
Las **Windows Forms** nos permiten crear aplicaciones ejecutables. Las **WPF** son aplicaciones también instalables en Windows, pero no son ejecutables como Windows Forms, sino XML apps similares, por ejemplo, a los gadgets utilizados en la barra lateral de Windows Vista y 7.
Las **aplicaciones de consola** se ejecutan en la línea de comandos. Con ellas, por ejemplo, podemos crear servicios de Windows.
Las **bibliotecas de clases** son archivos **.DLL** (*Dynamic Link Library*), que permiten interactuar a las aplicaciones creadas con cualquier lenguaje de programación, que se pueda ejecutar en Windows, a través de lo que se conoce como API (*Application Programming Interface*).
Las aplicaciones de **Explorador de Internet WPF** son programas iguales a las WPF, pero que corren desde un navegador web, como **Microsoft Internet Explorer**.
Todas estas aplicaciones tienen su limitación dentro de Windows, propia de una versión Express: no pueden conectarse a bases de datos remotas, el soporte para

crear clases es limitado y no es posible desarrollar programas comerciales (con valor monetario en el mercado), según el cluf.

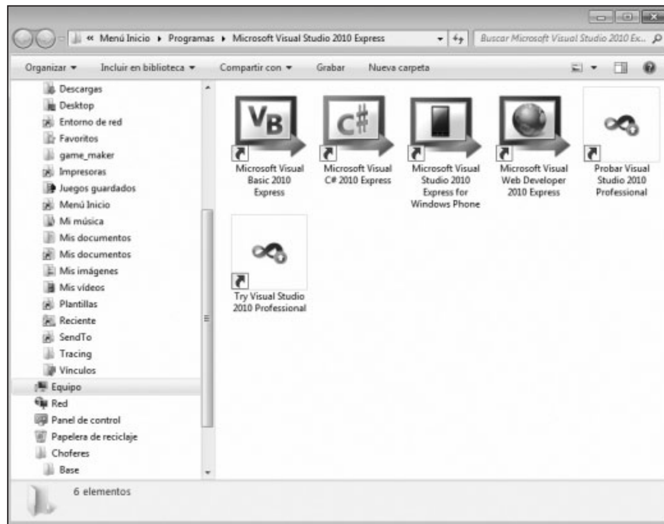


***Figura 11. Visual Studio 2010 Express** incluye los lenguajes de la suite en sus versiones reducidas y también ofrece probar la edición **Professional** durante 30 días antes de comprarla.*

Edición Standard

Es una edición paga de los lenguajes que componen la suite. Con ellos es posible crear el mismo tipo de aplicaciones que con las versiones Express, pero con alcances orientados al uso comercial. Incluye un mayor soporte para la creación de clases y permite crear instaladores del tipo **ClickOnce** para nuestros desarrollos.

Edición Professional

Brinda soporte a los componentes de las ediciones Standard y Express, y también incluye integración con SQL Server, al permitir crear las bases de datos desde el mismo entorno de Visual Basic. Ofrece depurador remoto de aplicaciones y desarrollo de programas para teléfonos móviles que corran **Windows Mobile** o el nuevo **Windows Phone** 7.



VERSIONES PARA ESTUDIANTES

Microsoft ha suscripto convenios con diversas universidades de todo el mundo, a partir de los cuales los estudiantes que se acrediten como alumnos regulares podrán adquirir la versión Professional de Visual Studio 2010 y SQL Server 2008 en forma gratuita. En **www.dreamspark.com** es posible verificar qué universidades figuran en la lista.

Edición Team System

Además de soportar todo lo mencionado para las ediciones anteriores, incluye herramientas colaborativas, destinadas a monitorear un equipo de desarrollo completo, presentar informes y gestionar las tareas de dichos equipos. También incluye un control de versiones conocido como **Team Foundation Server**.

Instalación de Visual Basic 2010

Luego de haber hecho un repaso por la historia de este fascinante lenguaje de programación y de haber visto sus características principales, prepararemos el terreno para instalar la aplicación en nuestra computadora y, así, comenzar a analizar en profundidad el lenguaje en sí.

El propósito de este libro es tomar la edición Express como base para aprender las características principales. Con ella, será más que suficiente para aprender el lenguaje y sus características más relevantes.

Cabe aclarar que todos los ejercicios que haremos con la edición Express pueden trasladarse a otras versiones superiores.

Instalación paso a paso

Desde la web oficial de **Microsoft** descargamos la versión Express del paquete **Visual Studio**. Podemos bajar solo **Visual Basic 2010**, pero dado que más adelante estudiaremos programación web mediante **ASP.NET**, precisaremos una herramienta adicional denominada **Visual Web Developer 2010**.

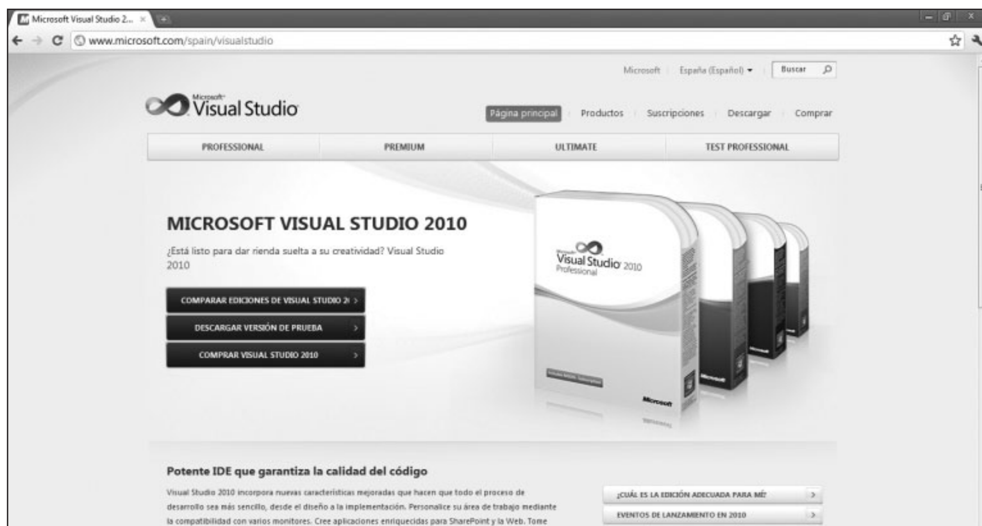


***Figura 12.** Desde el sitio oficial de Microsoft (**www.microsoft.com/express/Downloads**) podemos descargar la versión Express de Visual Studio 2010.*

Requisitos del sistema

Los requerimientos básicos que permiten correr las versiones Express o Professional de Visual Studio 2010 son:

- Procesador de 1,6 GHz o superior.
- Memoria RAM de 2 GB como mínimo para la versión Professional, y de 1 GB para la Express, sólo si se tiene XP como sistema operativo.
- De 4 a 7 GB de espacio en disco rígido, dependiendo de los componentes de la suite que elijamos instalar en nuestra computadora.
- Windows XP Service Pack 3, Vista, 7, Seven y para servidores Windows, desde Windows 2003 SP2 en adelante.

Descargar Visual Basic o Visual Studio 2010

Desde la web de Microsoft podemos descargar la versión Express de Visual Studio. De ella sólo estudiaremos Visual Basic 2010, con lo cual al iniciar el proceso, podemos optar por instalar únicamente Visual Basic si así lo deseamos. Nuestra recomendación es descargar e instalar la suite completa de Visual Studio, dado que, en el futuro, nos facilitará instalarla en otra computadora o tenerla a mano para reinstalar. Para hacerlo, de las opciones listadas en pantalla, elegimos **All - Offline Install ISO image file**. Luego de obtener la imagen de instalación en formato **.ISO**, con un programa grabador de DVD, creamos el disco de instalación. Una vez finalizado este paso, damos inicio a la instalación.

Iniciar proceso de instalación

Insertamos el disco de instalación en la unidad correspondiente de nuestro equipo y aguardamos unos segundos hasta que se abra la pantalla principal.
Desde el menú que se presenta, elegimos instalar dos de las aplicaciones listadas, que utilizaremos a lo largo de este libro: **Visual Basic 2010** y **Web Developer 2010**. Haciendo clic sobre Visual Basic 2010 se iniciará la instalación. Durante este proceso, seguramente se descargarán componentes adicionales desde Internet, con lo cual el tiempo de instalación promedio puede variar. Cuando termina esta etapa, procedemos a instalar **Visual Web Developer 2010**; esto será



DEMORAS EN LA INSTALACIÓN DE VISUAL BASIC 2010

Visual Basic 2010, como el resto de las aplicaciones que componen Visual Studio, requieren el framework .NET 4.0. Si este no fue instalado previamente, se descargará la versión necesaria desde la Web y se la instalará en la computadora, para recién luego continuar con la configuración del resto de los componentes.

mucho más rápido porque la mayoría de los componentes externos necesarios ya habrán sido instalados por Visual Basic 2010.

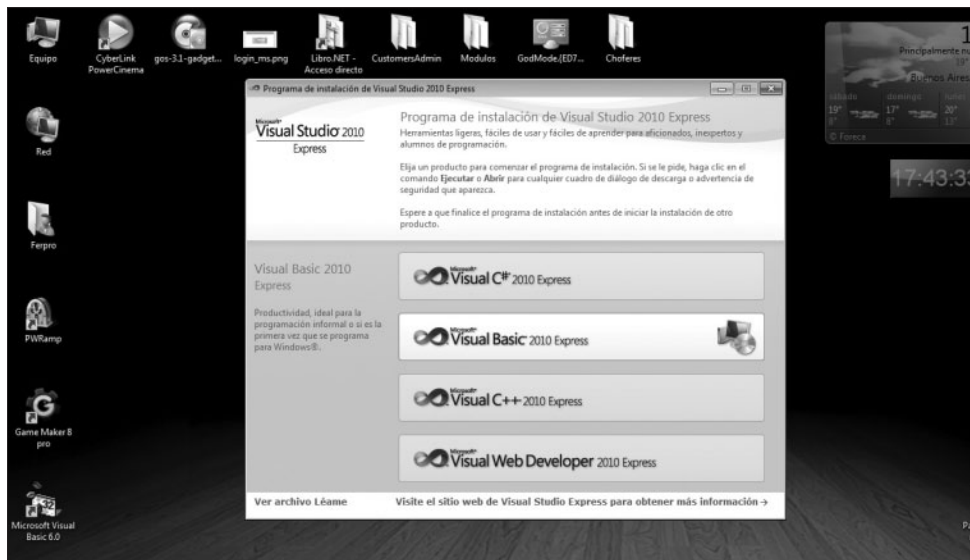


Figura 13. *El menú principal de Visual Studio 2010 desde donde podemos instalar todas las versiones gratuitas de la plataforma. Este menú es una aplicación de navegador creada con el propio Visual Studio.*

Dependencias

Para poder desarrollar los ejercicios que se presentan en este libro, dependeremos de herramientas adicionales que complementarán al lenguaje de programación. A continuación, detallamos cuáles son.

Versiones del framework .NET

En la actualidad, se encuentra en el mercado la versión 4.0 del framework .NET. Tanto Visual Basic 2010 como el resto de las herramientas que componen Visual Studio 2010 nos dejan trabajar solo con esta última versión. Visual Studio 2008 permitía interactuar con más de un framework, que podía elegirse al iniciar el proyecto, pero no se podía cambiar por una versión superior ni por una anterior una vez iniciado el desarrollo. A partir de esta versión de la suite, se brinda soporte a los frameworks 2.0, 3.0 y 3.5.

SQL Server Express / Professional

En el **Capítulo 5** introduciremos el concepto de base de datos y veremos cómo trabajar con ellas desde Visual Basic 2010. Para hacerlo, utilizaremos **SQL Server 2008 Express Edition**, que se instala junto con Visual Basic 2010.



***Figura 14.** Junto con la instalación completa de Visual Studio 2010 se instala la herramienta **Centro de instalación de SQL Server**. Desde ella podemos acceder a la ayuda, las actualizaciones y otros recursos para esta base de datos.*

La instalación de SQL Server 2008 que realizaremos corresponde solo al motor de base de datos. Para trabajar con la base de manera cómoda, necesitaremos instalar la interfaz gráfica que nos permita llevar a cabo esta tarea. **SQL Server 2008 Management Studio Express Edition** se puede descargar de manera gratuita desde la web de Microsoft: **www.microsoft.com/downloads/details.aspx?displaylang=es&FamilyID=08e52ac2-1d62-45f6-9a4a-4b76a8564a2b**.

Una vez descargado este componente, pasamos a instalar la aplicación. Con este último paso, ya disponemos en nuestra computadora de las herramientas necesarias para llevar adelante el aprendizaje de esta nueva edición de Visual Basic.

ENTORNO DE DESARROLLO

Quienes hayan trabajado con versiones anteriores a Visual Basic 2010 notarán que el entorno de desarrollo prácticamente no ha cambiado.



VERSIONES DE SQL SERVER

Tanto SQL Server como Visual Studio cuentan con versiones pagas y gratuitas; estas últimas, con ciertas limitaciones. Los ejercicios que realizaremos con bases de datos podrán llevarse a cabo tanto con la versión gratuita como con la paga de SQL Server 2008. También es posible utilizar la versión 2005 de esta base de datos.

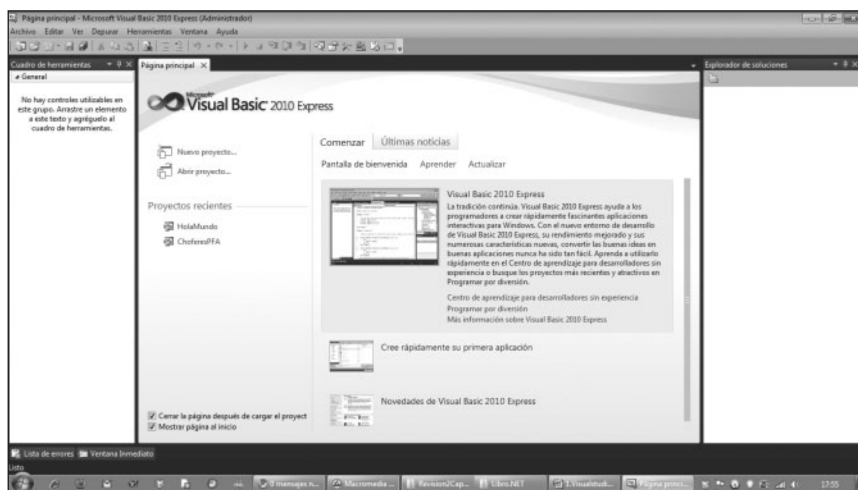


Figura 15. *Así es el entorno de desarrollo que ofrece Visual Basic 2010. Desde él podremos crear nuevas aplicaciones, abrir soluciones ya generadas, y estar al tanto de las últimas novedades provistas por el cliente RSS de Microsoft.*

Dentro del entorno de trabajo de Visual Basic 2010 encontramos la **Barra de menús**, la **Barra de herramientas**, el **Cuadro de herramientas**, el **Explorador de soluciones** y una pestaña denominada **Página principal**. Estos elementos son los necesarios para crear nuestros proyectos.

Comprender la estructura del IDE

Desde la página principal del **IDE** podemos iniciar un **Nuevo proyecto** o abrir un **Proyecto existente**, los cuales se detallarán en la lista de **Proyectos recientes**. En esta misma pestaña tendremos acceso a las webs preparadas por Microsoft, donde encontraremos información adicional sobre Visual Basic y Visual Studio, ejemplos rápidos para estudiar y otros beneficios adicionales, como video tutoriales y el canal **RSS** a través del cual podremos conocer las últimas novedades de Visual Basic y Visual Studio 2010, como así también los futuros **Services Packs** y versiones de prueba de nuevos productos.



LIBRERÍAS Y OBJETOS COM ANTIGUOS

Si disponemos de librerías OCX u objetos COM, DLL o demás componentes utilizados en versiones anteriores de Visual Studio .NET, podemos testear su compatibilidad agregándolos desde el Cuadro de herramientas. Solo tenemos que presionar el botón derecho del mouse y, del menú emergente, seleccionar **Elegir elementos**.

El sistema de ayuda

Visual Basic 2010 contiene un sistema de ayuda enlazado con la web de **MSDN**, a través del cual podremos acceder puntualmente a lo que necesitemos consultar.
Al iniciarlo por primera vez, Visual Basic 2010 nos notificará que es preciso elegir un sistema predeterminado para la ayuda. Si disponemos de conexión a Internet, siempre nos convendrá ver el contenido de ayuda de Internet en vez del sistema de ayuda local, ya que este último no tendrá actualizaciones inmediatas. La ayuda será visualizada en una pestaña dentro del IDE de trabajo de Visual Basic 2010.

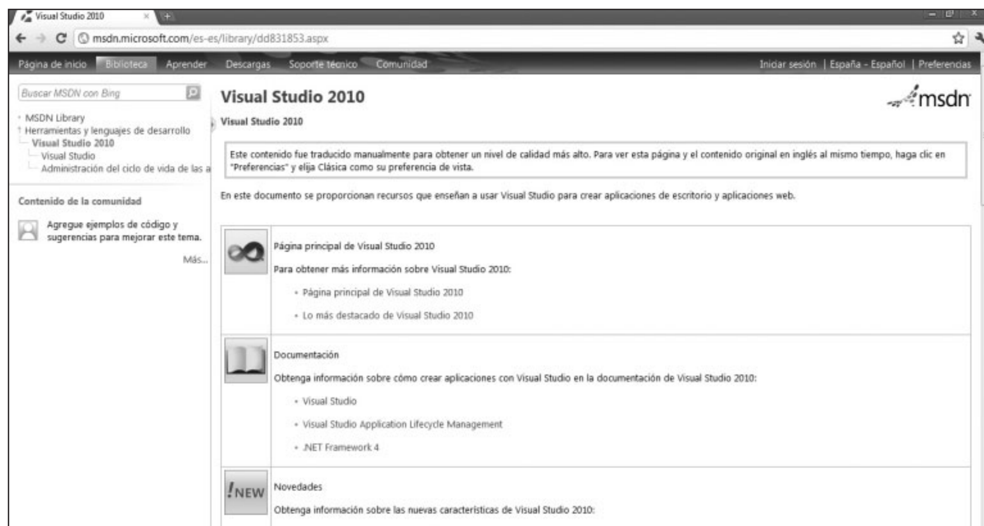


Figura 16. *Visual Basic dispone de un sistema de ayuda con el cual podemos consultar sobre cualquier herramienta o elemento del IDE para así evacuar nuestras dudas.*

Al comenzar a utilizar Visual Basic 2010, el sistema nos avisará que debemos registrarnos; aunque hayamos instalado la versión Express, tendremos que hacerlo. El proceso es rápido y gratuito. Una vez llevado a cabo este paso, recibiremos un correo electrónico en nuestra cuenta para que ingresemos en el IDE de Visual Basic y quedemos registrados.



VENTANAS Y SOLAPAS DEL IDE

El entorno IDE de Visual Basic 2010, como el resto de Visual Studio, es totalmente personalizable. Si queremos esconder el Cuadro de herramientas o el de propiedades, solo debemos realizar un clic en el pin superior de cada pestaña, y se ocultará. Para volver a verlo, posicionamos el mouse otra vez sobre dicha solapa.

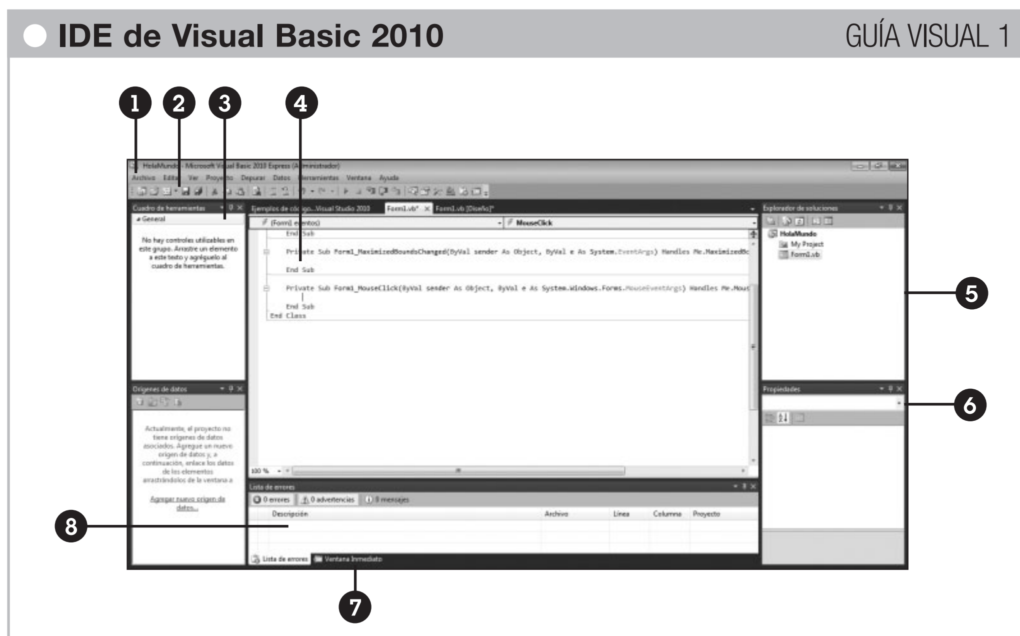


Figura 17. *Al registrar nuestro producto Visual Basic 2010, obtendremos la clave en la cuenta de correo electrónico ingresada. Con ella podremos instalar Visual Basic 2010 en otras máquinas sin necesidad de volver a registrar la copia Express.*

Comprender el IDE

Como mencionamos anteriormente, el IDE agrupa todo lo que necesitamos para trabajar dentro del entorno de desarrollo de Visual Basic 2010, por eso es sumamente importante conocerlo en detalle (**Guía Visual 1**). El hecho de familiarizarnos con el IDE de Visual Basic 2010 nos permitirá, en el futuro, saber cómo manejarnos con otros lenguajes de programación de la plataforma, dada la similitud con los otros lenguajes de Visual Studio 2010. También nos ayudará a que nuestros desarrollos sean más productivos y puedan realizarse en un corto tiempo.

IDE de Visual Basic 2010 — GUÍA VISUAL 1



❶ **Menús**: agrupa las funciones de personalización del IDE, la ayuda, la visualización de las ventanas contenedoras de herramientas, el acceso a propiedades, y la función de compilación de proyectos.
❷ **Barra de herramientas**: en ella se pueden agrupar los principales botones equivalentes a las funciones más utilizadas de los menús. La barra principal incluye las funciones de iniciar un nuevo proyecto, guardar, agregar uno o más objetos al proyecto, compilar y depurar, entre otras.
❸ **Cuadro de herramientas**: agrupa los controles que permiten diseñar la interfaz gráfica de las aplicaciones (botones, cajas de texto, grilla de datos, etc.). Algunos controles pueden no estar visibles al momento de ejecutar nuestro software.
❹ **Editor de código**: es el área de trabajo donde escribimos las sentencias que ejecutará el programa para funcionar.
❺ **Explorador de soluciones:** en él se listarán todos los archivos y recursos internos y externos que componen un proyecto.
❻ **Ventana de propiedades**: desde ella podemos asignar el nombre a formularios y controles, y ajustar las propiedades de cada uno de ellos, la posición en pantalla, las fuentes y los colores, entre otras características.
❼ **Ventana de inmediato**: aquí veremos en modo depuración los valores asignados a variables o a las propiedades de nuestros controles y objetos.
❽ **Lista de errores**: la lista de errores, advertencias y mensajes nos mantendrá informados sobre las equivocaciones cometidas al escribir el código y que el motor de depuración detecte. También permite evaluar datos de variables o espacios de nombre no declarados, entre otras advertencias.

Estructura de una solución

Visual Basic denomina solución a aquellos proyectos que iniciamos, y que guardarán la interfaz y el código fuente de una aplicación.
Dependiendo del tipo de proyecto de que se trate, la estructura de una solución puede llegar a variar. No es igual la de una solución **Windows Forms** que la de una



DESCARGA DE VISUAL BASIC 2010 EXPRESS EDITION

Si no queremos descargar la suite completa de Visual Studio 2010, podemos bajar solo **Visual Basic 2010 Express Edition** desde **www.microsoft.com/express/Downloads**. Se descargará a nuestro disco un simple ejecutable que comprobará los requisitos de la computadora y obtendrá todos los componentes adicionales necesarios para Visual Basic 2010.

solución **Aplicación de consola**. La primera contendrá **forms**, **módulos**, **archivo de recursos**, **módulos de clases** e **imágenes externas**, mientras que la segunda tendrá uno o más módulos, módulos de clase y recursos externos, pero no tendrá forms ni imágenes externas.

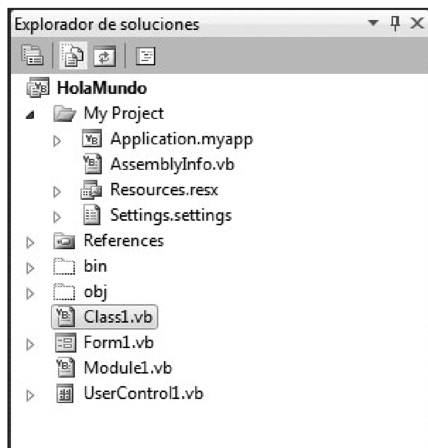


Figura 18. *En el* ***Explorador de soluciones*** *podemos ver todos los archivos que contiene nuestra solución, así como agregar y eliminar otros que necesitemos.*

Carpetas y archivos que componen una solución

Las carpetas y los archivos que componen una solución se guardan por defecto en una ruta específica dentro de **%profile%\Documentos\Visual Studio 2010\Projects**. Esta ruta puede cambiarse si necesitamos concentrar todos los proyectos en un único servidor de archivos o disco de red.

Si deseamos reorganizar dónde guardar nuestros proyectos antes de comenzar con los ejercicios del libro, este es el momento justo para modificar la ruta de acceso de nuestra carpeta. Para hacerlo, vamos al menú **Herramientas/Opciones**, marcamos **Mostrar todas las configuraciones** y, del panel izquierdo, seleccionamos **Proyectos/Soluciones/General** y allí especificamos la carpeta donde queremos almacenar los proyectos, como muestra la **Figura 19**.



MODIFICAR CÓDIGO SIN RECOMPILAR

A partir de la versión 2002 de Visual Basic se eliminó la posibilidad de cambiar código mientras se ejecutaba una aplicación. Esta característica permitió, hasta la versión 6.0 de Visual Basic, verificar las variables, corregir el código de las rutinas y volver a ejecutarlo sin necesidad de detener la aplicación e iniciarla otra vez.

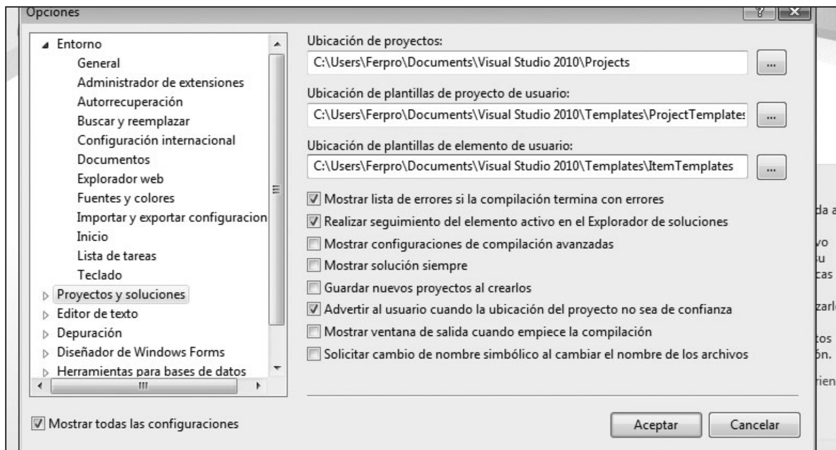


Figura 19. *Desde la ventana **Opciones** podemos personalizar a fondo el IDE de Visual Basic. Es ideal dedicarle un tiempo a esta herramienta para conocer bien todo lo que podemos cambiar en él.*

En la **Tabla 1** vemos una breve descripción de cada tipo de archivo o carpeta que compone una solución, con su función principal.

COMPONENTE	TIPO	DESCRIPCIÓN
BIN	Carpeta	Carpeta donde se compila el programa.
DEBUG	Carpeta	Carpeta donde se compila y depura el programa.
Resources	Carpeta	Carpeta donde se almacenan recursos externos, como archivos de imágenes.
Archivo.VB	Archivo	Archivos del tipo forms o módulos donde se almacena el código escrito junto con el diseño visual de la solución.
Archivo.PFX	Archivo	Archivo que genera un identificador único para la aplicación.
Archivo.SLN	Archivo	Solución de Visual Basic; es el archivo que contiene la información de todos los archivos y carpetas que componen un desarrollo.
Archivo.SUO	Archivo	Archivo que almacena la información de personalización del IDE realizada por el usuario.

Tabla 1. *Aquí podemos apreciar la extensión y función de cada archivo que compone la solución en Visual Basic 2010.*

Visual Basic agrupa las carpetas y los archivos pertenecientes a una solución dentro de un único lugar. Si bien podemos adicionar archivos externos, estos generalmente deben ser copiados de manera local a la carpeta contenedora de la solución, por una cuestión de seguridad. Podemos optar por dejar la ruta relativa al archivo que incorporemos a nuestra solución, pero Visual Basic siempre nos recomendará hacer una copia local.
Dentro de nuestra solución podemos crear subcarpetas para almacenar archivos en forma estructurada. Esto permitirá, por ejemplo, que proyectos de cientos de **forms** sean estructurados en la solución dentro de subcarpetas denominadas con el nombre del menú al que pertenece cada uno.

Podemos agregar nuevos archivos a nuestra solución actual, de manera muy práctica. Dichos archivos pueden crearse a partir de las plantillas que incluye Visual Studio 2010, con lo cual acortaremos más los tiempos de desarrollo y no tendremos que invertir media hora en diseñar una **pantalla de Login** o un cuadro **Acerca de**. Para incorporar un archivo a nuestra solución a partir de una plantilla, debemos dirigirnos a la solapa **Explorador de soluciones**, hacer clic derecho del mouse sobre el nombre de la solución y, en el menú, seleccionar **Agregar/Nuevo elemento**. Aparecerá una ventana que nos permitirá seleccionar entre varias opciones. En la **Figura 20** se muestran algunas de las disponibles.

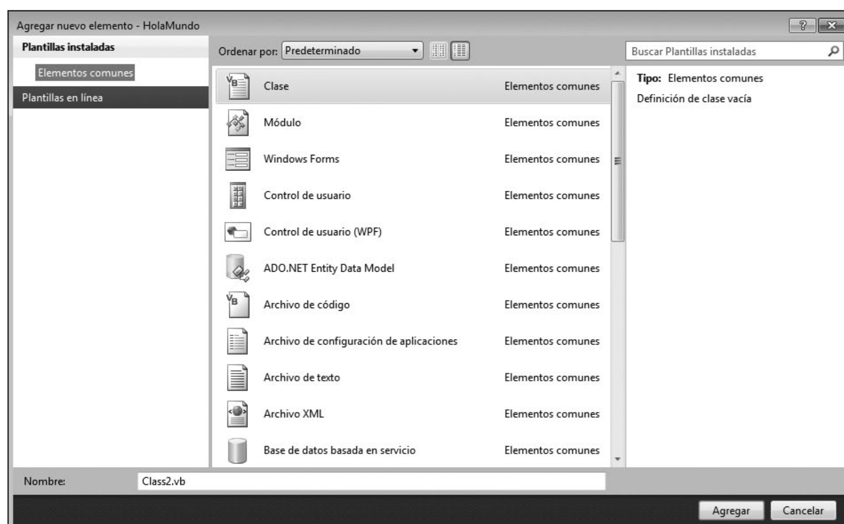


***Figura 20.** La ventana de plantillas permite acortar los tiempos de desarrollo y diseño, aprovechando desde un simple módulo de clase prediseñado o un Windows Form, hasta una base de datos basada en un servicio web.*

DESARROLLO DE NUESTRA PRIMERA APLICACIÓN

Hasta aquí hemos realizado un repaso por las características básicas que necesitamos conocer antes de iniciar un desarrollo. A partir de ahora, comenzaremos a usar el entorno de Visual Basic para llevar adelante nuestra primera aplicación, que involucrará algunas sentencias básicas sobre una solución del tipo **Windows Forms**.

Crear la solución

Para crear la solución que contendrá los archivos de nuestro primer programa, podemos optar por diferentes maneras de iniciarla: desde el menú **Archivo/Nuevo**

proyecto; desde el primer icono de la Barra de herramientas, **Nuevo proyecto**; presionando la combinación de teclas **CTRL + N**; o, al iniciar Visual Basic 2010, seleccionando la opción **Nuevo proyecto** de la Página principal. A continuación, veremos la ventana **Nuevo proyecto**, de donde seleccionamos **Aplicación de Windows Forms**.

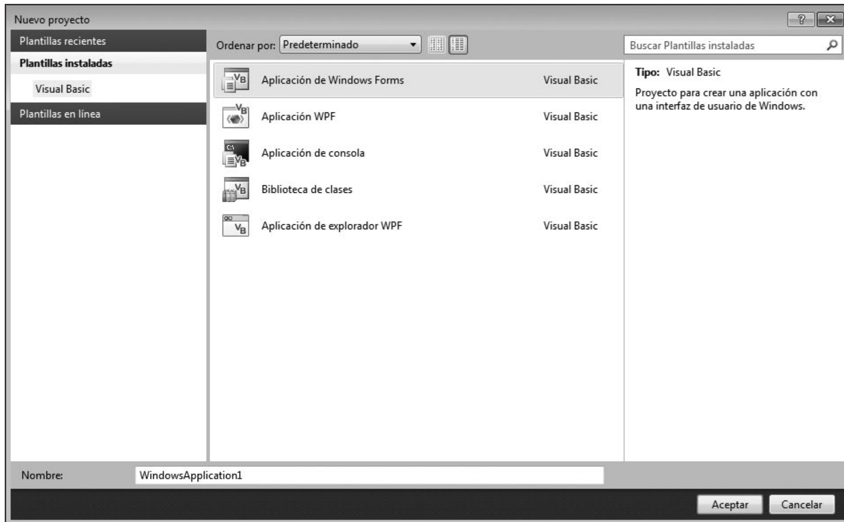
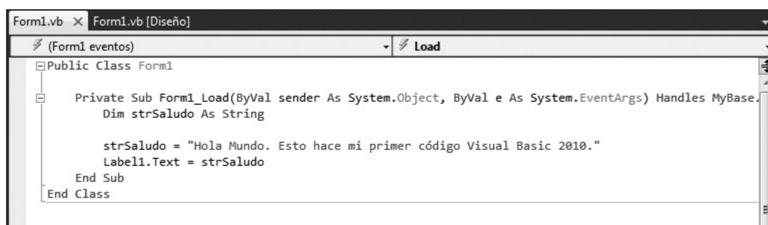


Figura 21. *Aquí podemos ver la ventana Nuevo proyecto y las diferentes opciones que nos ofrece Visual Basic 2010.*

La nueva aplicación que aparece en pantalla es una pestaña con un **Windows Form** vacío, listo para agregarle controles. En la pestaña **Explorador de soluciones** encontraremos el nombre de nuestra solución junto con dos archivos más: **My Project** y **Form1.VB**. Cambiamos en un principio el nombre de la aplicación, de **WindowsApplication1** a HolaMundo. Para esto, hacemos clic derecho del mouse sobre el título y, del menú contextual, seleccionamos **Cambiar nombre**.

Objetivo del primer proyecto

En nuestro primer proyecto vamos a usar un control **Label** y a escribir las primeras líneas de programación. El **Label** mostrará en la ventana principal la información que agregaremos a una variable interna del programa.




```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.
        Dim strSaludo As String

        strSaludo = "Hola Mundo. Esto hace mi primer código Visual Basic 2010."
        Label1.Text = strSaludo
    End Sub
End Class
```

Figura 22. *Nuestra primera solución codificada. El clásico "Hola Mundo".*

Codificación

Para incluir un **Label** en el programa, solo debemos buscarlo en el **Cuadro de herramientas** y arrastrarlo hacia el formulario **Form1**. Una vez realizado este paso, podemos cambiar el tamaño de la fuente por defecto del Label a uno mayor. Para esto, hacemos un solo clic sobre él y, en la ventana **Propiedades**, buscamos **Font** y **Forecolor**. En **Font** hacemos un clic sobre el lateral derecho de la propiedad y seleccionamos la fuente de nuestra preferencia, le damos un tamaño **16** y estilo **Negrita**. En la propiedad **ForeColor** desplegamos el combo mostrado sobre el lateral derecho y, de las opciones disponibles, elegimos la pestaña **Personalizado**. Allí seleccionamos un color, evitando el gris, ya que se perdería nitidez debido al color de **Form1**.

Una vez ajustadas las propiedades básicas de la solución, escribimos las primeras líneas de código que darán vida al programa. Hacemos doble clic en cualquier parte de **Form1**; se abrirá una pestaña con el siguiente código:

```
Public Class Form1

  Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

  End Sub
End Class
```

La primera línea de código **Public Class Form1** indica que Form1 es declarado como un objeto, y que cada control ubicado dentro de él será un objeto también.

Dentro de la clase **Form1** encontramos la sentencia **Private sub Form1_Load(...)**. Todo el código contenido en ella será ejecutado inmediatamente cuando se produzca el evento de carga de esta ventana.

Entre **Private Sub Form1_Load** y **End Sub** escribimos el siguiente código que dará vida a nuestra primera aplicación:

```
Dim strSaludo as string

Strsaludo = “Hola Mundo. Esto hace mi primer código Visual Basic 2010.”
Label1.text = strSaludo
```

Ejecución del proyecto

Antes de ejecutar la solución, debemos guardarla, para lo cual presionamos el botón **Guardar Todo** y, en la ventana que se presenta, ingresamos el nombre que le daremos al proyecto. Por defecto, se ofrece el nombre de proyecto igual a como

denominamos la solución. La ruta de guardado será la especificada de manera predefinida por Visual Basic 2010 o la que hayamos configurado nosotros mismos al editar las preferencias del entorno.
Para ejecutar nuestra primera aplicación, podemos presionar la tecla **F5** o ir al menú **Depurar/Iniciar depuración**. Así podremos apreciar nuestra aplicación en ejecución. Si bien este primer ejercicio es muy simple, al hacerla aprendimos los aspectos básicos que necesitamos tener en cuenta cuando escribimos código y la ubicación de las herramientas de Visual Basic IDE.

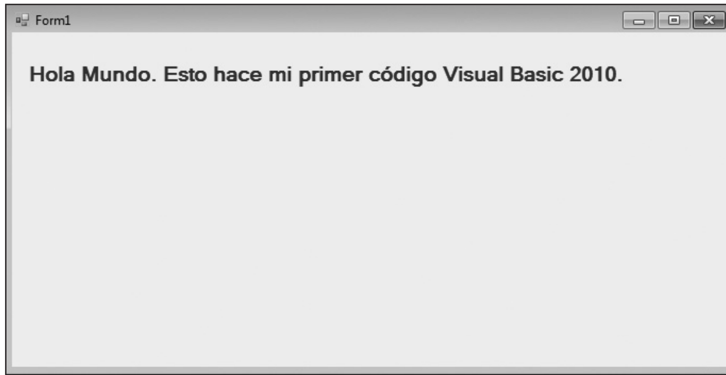


Figura 23. *Presionando la tecla F5, conseguimos ejecutar la aplicación, tal como si fuese un programa independiente.*

... RESUMEN

Hasta aquí hemos podido repasar brevemente la historia de este fascinante lenguaje de programación desde sus inicios en el antiguo sistema operativo DOS, y vimos cómo ha crecido con el tiempo, primero transformándose en una aplicación RAD de desarrollo de software para Windows, pasando por su adaptación en la versión 6.0 al mundo web, hasta convertirse en un verdadero lenguaje orientado a objetos desde que comenzó a llamarse .NET. También recorrimos sus distintas versiones, vimos cómo instalarlo y movernos en su entorno, y de qué manera realizar la primera aplicación.



ACTIVIDADES

PREGUNTAS TEÓRICAS

1. ¿Qué es Visual Studio?

2. ¿Qué versión del framework .NET se necesita para trabajar con Visual Basic 2010?

3. ¿Visual Basic 2010 permite integrar librerías VBX en el desarrollo de aplicaciones?

4. Nombre al menos tres tipos de aplicaciones que pueden desarrollarse con Microsoft Visual Basic 2010.

5. ¿Es necesario tener la versión Professional o superior de Visual Basic para desarrollar aplicaciones XAML?

6. ¿Hay que instalar toda la suite de Visual Studio 2010 para desarrollar software?

7. ¿Solo se pueden crear las aplicaciones de consola con la versión para DOS de Visual Basic?

8. El entorno de Visual Basic es personalizable por el usuario, y esta configuración es guardada en el archivo de extensión .VB. ¿Es esto correcto?

9. ¿Es posible en Visual Basic 2010 pausar el software en ejecución para modificar un fragmento de código sin recompilarlo?

10. El **CLR** agrupa las clases básicas que conforman el único **namespace** necesario para programar con cualquier lenguaje de Visual Studio. ¿Es esto correcto?

Fundamentos de Visual Basic

Realizaremos una introducción al concepto del lenguaje que luego nos acompañará en el desarrollo de todos los ejemplos de este libro. Conoceremos las palabras reservadas y los tipos de datos, utilizaremos variables dentro del sistema y veremos los diferentes tipos de declaraciones que podremos hacer. También trabajaremos con Windows Forms, repasaremos las instrucciones básicas del programa, y aprenderemos las principales diferencias entre los procedimientos y las funciones

CONCEPTOS DEL LENGUAJE

Desde el nacimiento de .NET, la plataforma de desarrollo Visual Basic ha evolucionado en pos de la programación orientada a objetos. Esto permite a los programadores centralizarse en el diseño de aplicaciones Windows, Web y las actuales basadas en XAML, y reducir de manera considerable la escritura de código para generar una UI (*User Interface*) amigable. Esto es, justamente, lo que hace que este lenguaje de programación haga honor a su nombre: **el desarrollo Visual**.
La integración de elementos prearmados conocidos como componentes, que se agrupan dentro del framework .NET, permite arrastrarlos y soltarlos sobre un formulario para así lograr una interfaz visual en apenas unos minutos. Luego, solo deberemos concentrarnos en codificar nuestra solución para que cumpla la función necesaria de todo programa: la interacción entre la computadora y el usuario final.

Cómo programar en Visual Basic

La computadora se ha transformado en una herramienta productiva que logró automatizar el funcionamiento de las empresas de manera notable. En el mundo actual de la programación, se busca constantemente acortar los tiempos de desarrollo, de la misma manera que una planilla de cálculo agilizó el proceso contable de un departamento de finanzas. Si bien hay muchos estilos diferentes de programación, lo primero que debe hacer el usuario es familiarizarse con las sentencias y las palabras que brinda el lenguaje, de modo de reducir los tiempos. Todo lenguaje de programación cuenta con algoritmos. Un **algoritmo** es la manipulación de números y ecuaciones que, a través de una serie de pasos organizados, nos permiten obtener la solución a un problema específico. Visual Basic 2010 incluye un conjunto de palabras reservadas y no reservadas que conforman símbolos y reglas para describir de manera explícita un proceso.



Figura 1. *Microsoft ofrece, en su sitio web, un completo fundamento sobre **POO** en **http://msdn.microsoft.com/es-es/library/bb972232.aspx**.*

Palabras reservadas

Si bien hasta ahora hicimos algún uso de las palabras reservadas, no hemos ahondado en el concepto en sí, y dado que a través de ellas resolveremos la mayoría de los problemas que se nos planteen en nuestros proyectos, es un buen momento para comprender mejor lo referente a éstas.

¿Qué son las palabras reservadas?

Las **palabras reservadas** tienen un significado gramatical específico para el lenguaje que estamos utilizando y no podrán ser empleadas como otro identificador del lenguaje. En ellas se agrupan **constantes**, **funciones**, **tipos de datos**, **variables** y **sentencias** que utilizaremos a lo largo de este libro. Con la práctica y su uso constante, terminaremos por aprenderlas y sabremos en qué momento debemos aplicarlas. El **CLI** (*Common Language Infraestructure*) del framework .NET contiene una especificación estandarizada necesaria para que nuestras aplicaciones funcionen correctamente.

Listado de palabras reservadas

Visual Basic 2010 cuenta con un listado de palabras reservadas y no reservadas, que se detallan en las **Tablas 1** y **2**.

PALABRAS RESERVADAS DEL LENGUAJE			
AddHandler	AddressOf	Alias	And
AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar
CDate	CDec	CDbl	Char
CInt	Class	CLng	CObj
Const	Continue	CSByte	CShort
CSng	CStr	CType	CUInt
CULng	CUShort	Date	Decimal
Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each
Else	ElseIf	End	EndIf
Enum	Erase	Error	Event
Exit	False	Finally	For
Friend	Function	Get	GetType
GetXMLNamespace	Global	GoSub	GoTo
Handles	If	If()	Implements
Imports (.NET)	Imports (XML)	In	Inherits
Integer	Interface	Is	IsNot
Let	Lib	Like	Long
Loop	Me	Mod	Module

PALABRAS RESERVADAS DEL LENGUAJE			
MustInherit	MustOverride	MyBase	MyClass
Namespace	Narrowing	New	Next
Not	Nothing	NotInheritable	NotOverridable
Object	Of	On	Operator
Option	Optional	Or	OrElse
Overloads	Overridable	Overrides	ParamArray
Partial	Private	Property	Protected
Public	RaiseEvent	ReadOnly	ReDim
REM	RemoveHandler	Resume	Return
SByte	Select	Set	Shadows
Shared	Short	Single	Static
Step	Stop	String	Structure
Sub	SyncLock	Then	Throw
To	True	Try	TryCast
TypeOf	Variant	Wend	UInteger
ULong	UShort	Using	When
While	Widening	With	WithEvents
WriteOnly	Xor	#Const	#Else
#ElseIf	#End	#If	=
&	&=	*	*=
/	/=	\	\=
^	^=	+	+=
-	-=	>> (Operador)	>>= (Operador)
<<	<<=		

Tabla 1. *Las palabras clave detalladas en esta tabla no pueden ser utilizadas como nombres de otros elementos de programación, por ejemplo, variables o procedimientos.*

PALABRAS NO RESERVADAS			
Add	Ansi	Assembly	Auto
Binary	Compare	Custom	Distinct
Equals	Explicit	From	Group By
Group Join	Into	IsFalse	IsTrue
Join	Key	Mid	Off
Order By	Preserve	Skip	Skip While
Strict	Take	Take While	Text
Unicode	Until	Dónde	#ExternalSource
#Region			

Tabla 2. *Las palabras aquí listadas no están reservadas y pueden utilizarse como nombres para otros elementos, aunque se recomienda no hacerlo, para evitar complicaciones en la lectura del código.*

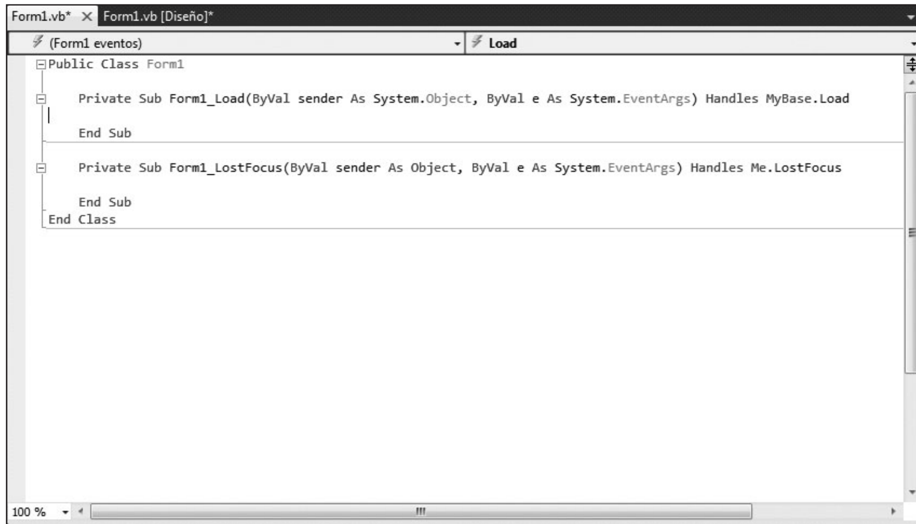


***Figura 2.** En la imagen podemos ver, dentro del código del Form1, algunas de las palabras reservadas, como **Private**, **Sub**, **ByVal**, **As**, **Handles** y **End Sub**, entre otras.*

Tipos de datos

Visual Basic clasifica los tipos de datos en dos grupos principales: los **Tipos por valor** y los **Tipos por referencia**. Los primeros almacenan datos que pueden ser accedidos de manera directa, y que, a su vez, se organizan en subgrupos: los **Tipos de datos nativos de .NET**, los **Tipos de datos creados por el programador** y los **Tipos de datos enumerados**. Visual Basic 2010 permite emplear la mayoría de los tipos de datos que estaban disponibles en sus versiones anteriores, salvo casos como el tipo de datos **Variant**, que dejó de existir cuando Visual Basic se transformó en un lenguaje orientado a objetos, y fue reemplazado por **Object**. Podemos entender esto mejor a través de algunos ejemplos.

El tipo de datos **Object** proviene del espacio de nombres **System.Object**, y en él se puede almacenar cualquier tipo de valor. Como este, el resto de los tipos de datos básicos se encuentran definidos dentro del namespace **System**, lo que hace que hereden directamente de **System.Object**.



PALABRAS RESERVADAS EN DESUSO

Las palabras **Endif**, **GoSub**, **Let**, **Variant** y **Wend** se conservan como palabras clave reservadas por una simple cuestión de retrocompatibilidad con las versiones anteriores del lenguaje, aunque realmente ya no son utilizadas en Visual Basic.

Vamos a iniciar Visual Basic 2010 y a crear a continuación un **Nuevo proyecto** del tipo **Aplicación de Windows Forms**, al cual llamaremos **TiposDeDatos**. En el primer código que escribimos, el programa simplemente crea dos tipos de datos distintos y les asigna valores. Para esto, hacemos doble clic en el formulario **Form1**, y dentro del procedimiento **Form1_Load** escribimos lo siguiente:

```
Dim oNac As Object
Dim strValor As String
'Asignamos una cadena de texto al tipo de datos strValor
strValor = "VISUAL BASIC 2010"
'Asignamos una fecha al tipo de datos oNac
oNac = "09/07/1926"
'Asignamos una cadena de texto al tipo de datos oNac
oNac = "Julio"
```

En las primeras dos líneas, declaramos los tipos de datos **Object** y **String**, asignándoles un nombre distinto a cada uno: **oNac** y **strValor**, respectivamente. Las líneas que tienen un apóstrofo delante son los comentarios que podemos realizar dentro del programa, que no serán interpretados por el compilador al momento de ejecutarse. Luego, definimos un valor para **strValor** y otros dos para **oNac**. En este último caso, el valor que quedará asignado a **oNac** será el último, ya que reemplaza al primero. Ejecutamos la aplicación presionando la tecla **F8**. Esta manera nos permitirá seguir el código paso por paso, y así ir viendo, al momento, cómo cada variable va tomando un valor. El cursor de la ejecución del programa se detendrá en la primera sentencia del código, la resaltará en color amarillo e indicará en el borde izquierdo con una flecha amarilla qué línea se está procesando.

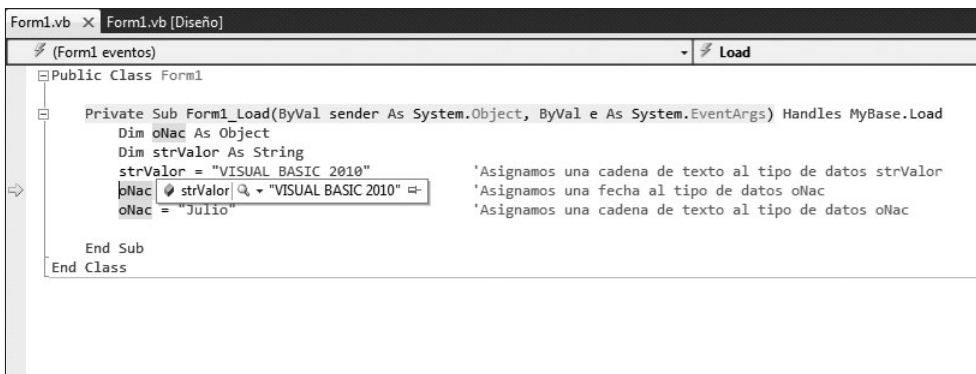


Figura 3. *Al seguir paso a paso el código de nuestro programa, podemos ver cómo la línea en proceso se resalta con una flecha amarilla lateral, en tanto que nuestros comentarios se destacan en verde.*

Para proseguir, debemos presionar **F8** reiteradas veces. Una vez que el cursor de ejecución se encuentra en la línea **oNac = "09/07/1926"**, acercamos el puntero del mouse hacia **strValor**. Veremos que nos muestra el dato asignado a él. Seguimos procesando las líneas de código hasta **End Sub** y volvemos a revisar el tipo de dato **oNac**. Veremos que el valor almacenado en él es la cadena de texto **"Julio"**.

Una variable **Long** declarada en Visual Basic almacena un valor entero de **64 bits**; por el contrario, una variable **Integer** almacena valores enteros de **32 bits**. Desde la aparición de .NET se incluyeron nuevos valores de enteros, como **Short**, que almacena valores de **16 bits**, y **Byte**, que almacena valores de **8 bits**, aunque solo números positivos. Agreguemos algunas líneas de código más a nuestro proyecto para ver cómo actúan estas variables, incluso, si les asignamos un valor que no soportan.

```
'Números enteros positivos desde 0 a 255
Dim b As Byte = 255

'Números enteros desde -2.147.483.648 a 2.147.483.647
Dim i As Integer = 2147483647

'Números enteros desde -9.223.372.036.854.775.808 a (9.2E+18)
Dim l As Long = 9223372036854775807
```

Ejecutamos lo escrito presionando **F8**. Al llegar a estas líneas, veremos cómo cada variable obtiene su valor. Los valores asignados son los máximos soportados por cada tipo de datos declarado. Si agregamos a cualquiera de ellos un dígito más, el programa provocará una excepción no controlada y notificará que el tipo de datos no soporta el valor asignado. Tal vez, quienes ya tienen conocimiento en programación esto les parezca políticamente no correcto, pero en un análisis detenido, puede revelar que utilizando correctamente cada nuevo tipo, en las aplicaciones que manipulan muchas variables numéricas, se obtendrá una gran diferencia en la memoria consumida por la aplicación, dado que se logrará una optimización significativa.



COMENTARIOS DENTRO DEL CÓDIGO

Siempre es bueno comentar el código escrito de nuestra aplicación. De esta manera, lograremos encontrar rápidamente la sentencia que necesitamos cuando nuestro programa ha crecido y contiene cientos o miles de líneas. Es importante tener esto en cuenta y hacer aunque sea un breve comentario en lugares donde se superan las diez líneas de código.

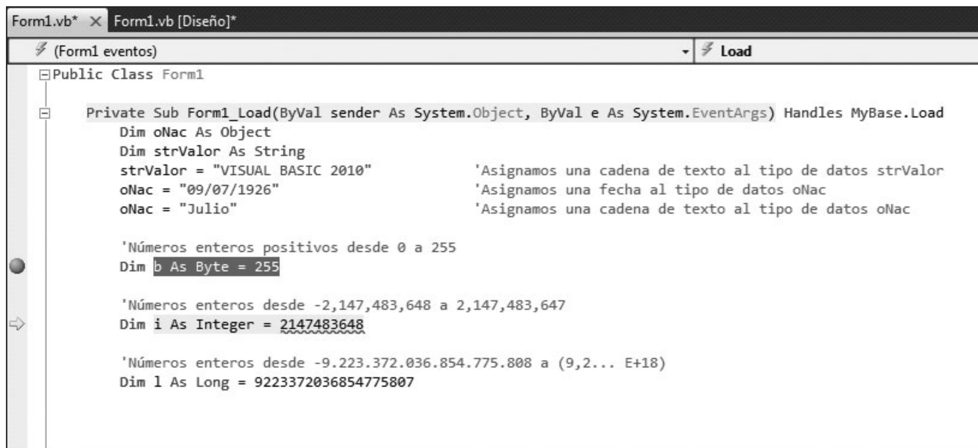


Figura 4. *Al tipo de datos **Integer** le asignamos un número más del máximo soportado. Visual Basic 2010 detecta este problema y, mediante la **Lista de errores**, notifica que el tipo de datos **Integer** no puede ser representado.*

Veamos ahora qué sucede si invertimos la asignación de datos en distintos tipos de variables. Vamos a escribir el siguiente código al final de todo lo anterior, y ejecutamos el programa hasta estas líneas:

```
Dim l1 As Long
Dim s2 As String

l1 = 12345
s2 = “Nicolas”
```

Ahora cambiamos algunos valores y ejecutamos otra vez el programa con **F8**:

```
S2 = 12345
```

La variable **s2**, que es un tipo de datos **String**, permite almacenar un valor numérico, dado que soporta letras, números y caracteres de toda clase. Ahora cambiamos el valor **l1**, que es del tipo **Long**, por una cadena de texto:

```
l1=”Nicolas”
```

Al ejecutar la aplicación, surgirá un error que nos indica que el tipo de datos **l1** no puede almacenar una cadena de texto.

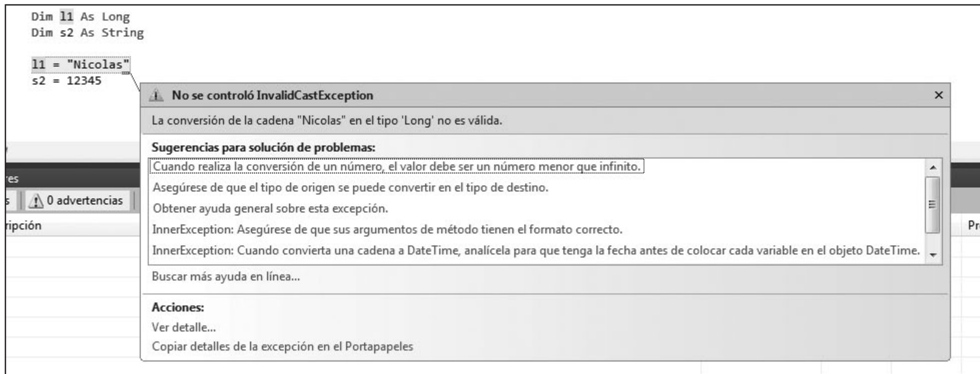


Figura 5. *Visual Basic nos avisa del error que cometimos a través de un cuadro de diálogo, y hasta nos sugiere desde allí consultar la ayuda y las posibles soluciones para resolverlo.*

Otros tipos de datos que utilizaremos a lo largo de este libro son **Boolean**, **Decimal**, **Single** y **Double**, entre otros. El primero nos permite guardar un estado Verdadero o Falso. En el código que se presenta a continuación realizamos una comparación para saber si una variable tiene un determinado valor, y sobre la base de ese resultado, asignamos **bResp** como verdadero o falso:

```
Dim bResp As Boolean, a As Long
a = 1234
If a = 1234 Then bResp = True Else bResp = False
'Si a = 1234 entonces bResp es verdadero, caso contrario bResp es falso
MsgBox(bResp.ToString)
```

Existen muchos más tipos de datos, que iremos conociendo mejor a lo largo del libro. Por el momento, los mencionaremos de manera generalizada para tenerlos presentes: **Byte** consume valores de **1 byte**; **Char** ocupa **2 bytes** y soporta 65536 caracteres; **Date** ocupa **8 bytes** y almacena formatos de fechas entre 1 de enero de

DECLARACIÓN DE VARIABLES

Cuando tenemos que declarar una variable, podemos hacerlo al inicio del evento, al inicio del Form o dentro de un módulo, y más adelante, asignarle un valor. Pero también podemos, en una misma línea de código, declararla y asignarle su valor si es que ya lo sabemos.

0001 y 31 de diciembre de 9999; **Double** ocupa **8 bytes** de memoria y permite almacenar números de punto flotante con precisión de **64 bits**; **Long** ocupa 4 bytes de memoria y almacena números que van de -2 mil millones hasta más de 2 mil millones; **Sbyte** ocupa **1 byte** de memoria y maneja valores desde -128 hasta 127; **Short** ocupa **2 bytes** de memoria y maneja valores desde -32.768 hasta 32.767; **Singl**e ocupa **4 bytes** de memoria y permite manejar números de punto flotante de precisión sencilla de **32 bits**; y **Decimal** ocupa **16 bytes** de memoria y permite almacenar números de formato de coma fija, y manipular valores de manera óptima sin problema de redondeos en un rango de **128 bits**. Cada uno de estos tipos de datos hereda sus propiedades y métodos de manera implícita del tipo **Object**. El framework .NET nos facilita el hecho de memorizar los valores mínimos y máximos soportados por cada uno de estos tipos de datos. Para saber si un valor por aplicar está fuera o no del rango permitido por el tipo de dato, podemos hacerlo de la siguiente manera:

```
Dim sNumero as single
sNumero.MinValue ‘nos devuelve el valor mínimo
SNumero.MaxValue ‘nos devuelve el valor máximo
```

Uso de variables

Las variables son espacios de memoria reservados para alojar un determinado tipo de valor que será utilizado durante la ejecución de una aplicación. Dicho valor almacenado puede cambiar si es necesario, en cualquier momento que lo dispongamos. Las variables también cuentan con un modificador de acceso, que determina su alcance dentro de nuestro proyecto. Veamos a continuación un detalle del uso de variables a través de la **Tabla 3**.

MODIFICADOR	DESCRIPCIÓN
Dim	Indica que la variable puede ser usada dentro de una estructura de control o clase donde haya sido creada.
Private	La variable podrá ser usada en una clase o módulo donde fue creada.
Public	La variable será accesible dentro de la solución donde fue declarada.
Friend	Este modificador es similar al anterior, y se restringe solo al proyecto al que pertenece.

***Tabla 3.** Aquí vemos los distintos tipos de variables que pueden declararse dentro de **Visual Basic 2010**.*

La creación de una variable local puede realizarse al principio del procedimiento, función o evento, o inmediatamente antes del momento de usarla. Iniciemos un nuevo proyecto del tipo **Windows Form**, agreguemos al **Form** dos controles **Button**, y escribamos en el evento **Click** de **Button1** el siguiente código:

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim sNombre As String = "Julián"
    MsgBox(sNombre)
  End Sub
```

Luego, ejecutemos el programa presionando **F5** y pulsemos **Button1**. Obtendremos un cuadro de mensaje con el valor asignado a la variable.

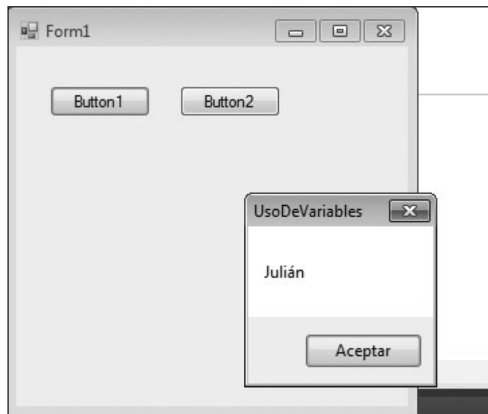


***Figura 6.** El cuadro de diálogo muestra el valor contenido en nuestra variable sNombre.*

Detengamos el proyecto, y ahora, en **Button2_click()** agreguemos el siguiente código y ejecutemos otra vez la aplicación:

```
MsgBox(sNombre)
```

Por un lado, al momento de escribir el código, Visual Basic nos está informando, mediante la Lista de errores, que la variable **sNombre** no fue declarada dentro de



NOMBRES DE LAS VARIABLES

Todo nombre de variable debe comenzar con una letra, no con un número, y tampoco pueden quedar espacios intermedios dentro de él. Cuando hay un equipo de programadores, es una buena práctica asignar al nombre, como primera letra, la misma del tipo de datos que vamos a declarar. Por ejemplo, **Dim strNotas as String**, **sTotal as single**.

Button2. Entonces, nuestro código no se podrá ejecutar. Igualmente, el compilador de Visual Basic se dará cuenta del problema de sintaxis y nos dará algunas posibles sugerencias para subsanarlo antes de pasar a la ejecución. Esto permite a todo desarrollador minimizar la posibilidad de errores al momento de compilar definitivamente una aplicación.

Para utilizar la variable en ambos botones, y que esta muestre su valor, debemos declararla al inicio de la clase **Form1**. Vamos a cortar el código correspondiente a la creación de la variable y a pegarlo al principio de la ventana de código:

```
Public Class Form1
Dim sNombre As String = "Julián"
…
```

Ahora, si ejecutamos el proyecto, veremos que presionando **Button1** o **Button2**, se mostrará el cuadro de mensaje. Esto demuestra que toda variable creada al inicio de cualquier clase **Form** permitirá disponer de su valor para cualquiera de los controles u objetos creados dentro del programa.



Figura 7. *Al dimensionar la variable de manera Pública dentro de nuestra clase Form1, podemos acceder a ella desde cualquiera de sus objetos.*

COMPILADOR DE VISUAL BASIC

En Visual Basic, un **Analizador de sintaxis** se ejecuta en segundo plano todo el tiempo, y revisa constantemente el código que escribimos. Por ejemplo, si llega a detectar un error tan común como la asignación de un valor a una variable que aún no fue creada, nos dará algunas sugerencias para corregirlo y no permitirá su ejecución.

Convertir tipos de datos

El uso de variables nos permite manipular datos almacenados en ellas. Si las variables son de tipo numérico, podremos realizar cualquier clase de cálculos entre ellas. Si son del tipo texto, podremos cambiar los mismos a mayúsculas o minúsculas, entre otras acciones más. El valor de las variables también puede representarse en cualquier control que tengamos en pantalla.

Creemos un nuevo proyecto **Windows Form** llamado **ConversionDeDatos**, y al inicio de la clase **Form1** declaramos las variables tal como figuran en el siguiente código para una mejor comprensión:

```
Public Class Form1
   Dim iUno As Integer = 15
   Dim iDos As Integer = 39
   Dim iResult As Integer
…
```

A continuación, dentro del evento **Load** de **Form1** escribimos lo siguiente:

```
   iResult = iUno + iDos
```

Ejecutamos nuestro proyecto pulsando **F8** y, una vez que el modo **DeBug** pasa la operación que almacena el resultado en la variable **iResult**, posicionamos el cursor del mouse sobre ella. Veremos allí el resultado obtenido en la suma de los valores contenidos en las variables **iUno** e **iDos**.

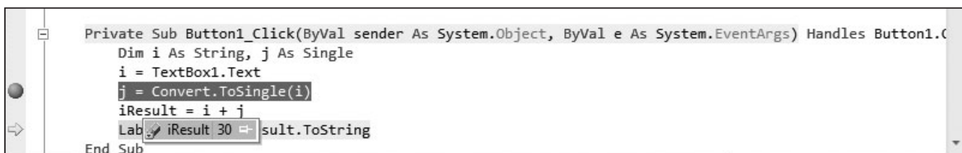


***Figura 8.** El inspector de elementos nos permite, en modo de depuración, ver los valores que poseen nuestras variables.*

A continuación, eliminamos el código escrito en el evento **Load** de **Form1**. Volvemos a la vista de diseño de **Form1** y agregamos dos **textBox**, un **Button** y un **Label**. Al finalizar, añadimos en **Form1_Load** el siguiente código:

```
TextBox1.Text = iUno
TextBox2.Text = iDos
```

Y en el evento **Click** de **Button1** agregamos el siguiente código:

```
Label1.Text = TextBox1.Text + TextBox2.Text
```

Ejecutamos la aplicación. Veremos que en **TextBox1** y **TextBox2** automáticamente aparecen los valores asignados a **iUno** e **iDos**. Presionamos el botón para ver si se realiza la suma y se muestra el resultado en **Label1**.
En vez de obtener el resultado de la suma de valores, en **Label1** vemos el contenido de ambas variables de manera consecutiva: eso se llama concatenación. El problema ocurre porque nunca dijimos a nuestra aplicación que los valores cargados en los **TextBox** son numéricos, y que deben sumarse y entregar como resultado el total.
Para llevar a cabo esto, modificamos el contenido del **Button1_Click** por lo siguiente:

```
Label1.Text = CInt(TextBox1.Text) + CInt(TextBox2.Text)
```

Ejecutamos otra vez nuestro programa y veremos que, en **Label1** obtenemos la suma de **textBox1** y **textBox2**. Esto que realizamos se denomina **conversión de tipos**. Es necesario especificar que el contenido de ambos **textBox** es **numérico** y que necesitamos realizar una operación matemática sobre ellos para obtener un resultado. Si esto no es llevado a cabo, interpretará que queremos realizar una concatenación.
Para especificar que un valor sea convertido a otro podemos usar **cInt()**; es parte de una serie de funciones que se mantienen dentro del lenguaje por una cuestión de retrocompatibilidad. En la actualidad, el **framework .NET** nos provee de una función nueva para realizar las conversiones de tipos: **Convert**. En la **Tabla 4** vemos las diversas opciones que ofrece esta función.

CONVERT	VALOR DEVUELTO	ARGUMENTO DE LA EXPRESIÓN
ToBoolean	Boolean	Convierte una expresión a verdadero o falso.
ToChar	Char	Convierte una expresión a Char.
ToSByte	SmallByte	Convierte una expresión a SmallByte.
ToInt16	Int16	Convierte una expresión a Int16.
ToInt32	Int32	Convierte una expresión a Int32.
ToInt64	Int64	Convierte una expresión a Int64.
ToUInt16	UInt16	Convierte una expresión a UInt16.
ToUInt32	UInt32	Convierte una expresión a UInt32.
ToUInt64	UInt64	Convierte una expresión a UInt64.
ToSingle	Single	Convierte una expresión a Single.
ToDouble	Double	Convierte una expresión a Double.
ToDecimal	Decimal	Convierte una expresión a Decimal.

CONVERT	VALOR DEVUELTO	ARGUMENTO DE LA EXPRESIÓN
ToDateTime	DateTime	Convierte una expresión a DateTime.
ToString	String	Convierte una expresión a String.

***Tabla 4.** Aquí vemos las distintas opciones de conversión que nos ofrece el **namespace** System.Convert, aunque también existen restricciones entre la conversión inicial y la final.*

La conversión de tipos de datos puede generar hasta cinco resultados distintos, dependiendo de su valor base inicial y del valor base final:

1. **Ninguna**: ocurre cuando el origen y el destino tienen el mismo tipo de dato. Por ejemplo, aplicar **Convert.ToString** a una variable del tipo **string**.
2. **InvalidCastException**: ocurre cuando no se admite una conversión determinada. Las conversiones no admitidas por los tipos de datos son:
   - Origen: **Char**, Destino: **Boolean**, **Single**, **Double**, **Decimal**, **DateTime**.
   - Origen: **Boolean**, **Single**, **Double**, **Decimal**, **DateTime**, Destino: **Char**.
   - Origen: **DateTime**, Destino: cualquier tipo exceptuando **String**.
   - Origen: Cualquier otro tipo excepto **String**, Destino: **DateTime**.
3. **FormatException**: ocurre cuando un valor de cadena no se puede convertir a ningún otro tipo base porque la cadena no tiene un formato correcto. Por ejemplo, sería imposible convertir a **Boolean** un **String** que no sea True o False.
4. **Conversión correcta**: ocurre cuando intentamos convertir algo que no produce pérdida de datos. Por ejemplo, un **String** a otro **String**.
5. **OverFlowException**: ocurre cuando convertimos un origen a un destino donde se produce una pérdida de datos. Por ejemplo, si tenemos una variable **Single** con un valor asignado **1000000** e intentamos convertirla a **Byte**, se producirá el **OverflowException**, ya que **Byte** solo acepta hasta **127**.

A continuación, vamos a modificar nuestro proyecto **ConversionDeDatos**. Borramos el contenido de **Button1_Click** y agregamos el siguiente código:

```
Dim i As String, j As Single
i = TextBox1.Text
j = Convert.ToSingle(i)
```



FUNCIONES DE CONVERSIÓN

Muchas funciones de conversión de Visual Basic han sido incluidas por retrocompatibilidad con la migración de proyectos hacia la última versión del lenguaje, pero en algún momento dejarán de tener soporte. Se recomienda que, al ir conociendo mejor el lenguaje, aprendamos en qué namespace se alojan para usarlas en proyectos nuevos.

Luego, presionamos **Button1** y esperamos a que el compilador lleve a cabo la conversión codificada. Veremos que al ejecutarse la misma ocurrirá un error de conversión de tipos no contemplado hasta el momento.

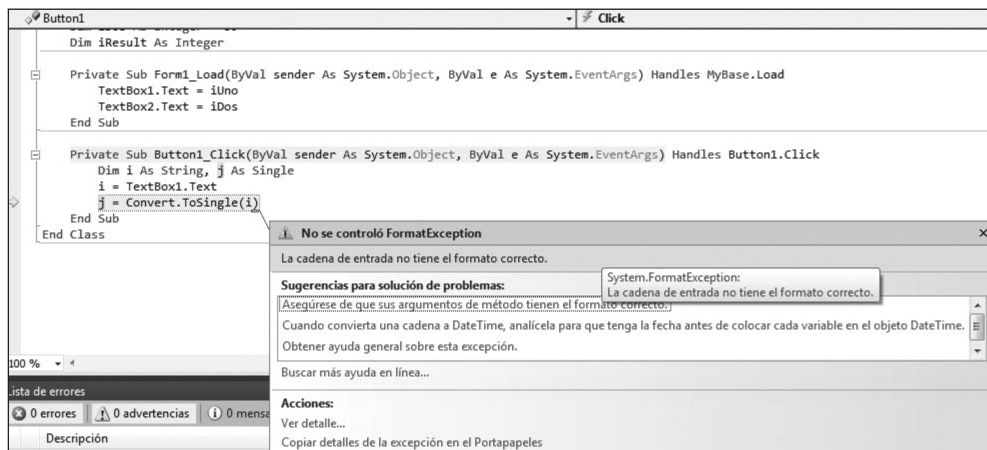


***Figura 9.** El error FormatException. En este ejemplo, vemos uno de los errores que se producen al intentar convertir un tipo String al tipo Single.*

Arrays y enumeraciones

Dentro de la denominación de las variables, en determinadas ocasiones podemos detectar que una misma declaración puede servirnos para almacenar mucha información correspondiente a un dato. Para lograr esto utilizaremos los arrays. Un array es una lista indexada de datos que puede contener mucha información. La manera correcta de declararlo es la siguiente:

```
Dim ElementosDeCocina(8) as string
```

Con este ejemplo, declaramos un **array** de una dimensión que puede alojar hasta 9 elementos de tipo **string**. Si bien su dimensionado es 8, tanto en **.NET** como en la mayoría de los lenguajes de programación el primer índice comienza desde **0** y no, en **1**. La declaración de arrays estáticos, deben utilizarse por completo para no desperdiciar los espacios de memoria reservados para éstos.
Iniciamos un nuevo proyecto **Windows Form** llamado **ManejoDeArrays**. Antes de la declaración **Public Class Form1** agregamos la siguiente línea:

```
Imports System.Collections
```

Esto añade el **Namespace System** al proyecto e incluye también la clase **Collections**. Dentro de la clase **Form1** declaramos **ElementosDeCocina(8)**. Luego, en el evento **Form1_Load** escribimos lo siguiente:
Como mencionamos antes, el array declarado aquí es un array estático, el cual deberá ser utilizado (dado sus 9 posiciones declaradas), para ser llenado por completo. Aunque no es un requerimiento estricto llenar por completo un array, siempre conviene hacerlo para evitar posibles errores en operaciones que realicemos con cada ítem almacenado.

```
ElementosDeCocina(0) = "Mueble"
ElementosDeCocina(1) = "Cocina"
ElementosDeCocina(2) = "Cubiertos"
ElementosDeCocina(3) = "Pileta"
ElementosDeCocina(4) = "Cafetera"
ElementosDeCocina(5) = "Platos"
ElementosDeCocina(6) = "Especias"
ElementosDeCocina(7) = "Aderezos"
ElementosDeCocina(8) = "Grill"

For i = 0 To UBound(ElementosDeCocina)
MessageBox.Show(ElementosDeCocina(i))
Next
```

En la misma línea de **ElementosDeCocina(8)**, hacemos un clic sobre el borde izquierdo de la ventana de código. Aparecerá un punto rojo y el texto se resaltará en rojo también. Presionamos a continuación **F8** y seguimos paso a paso el código para ver cómo se cargan los datos en el array.

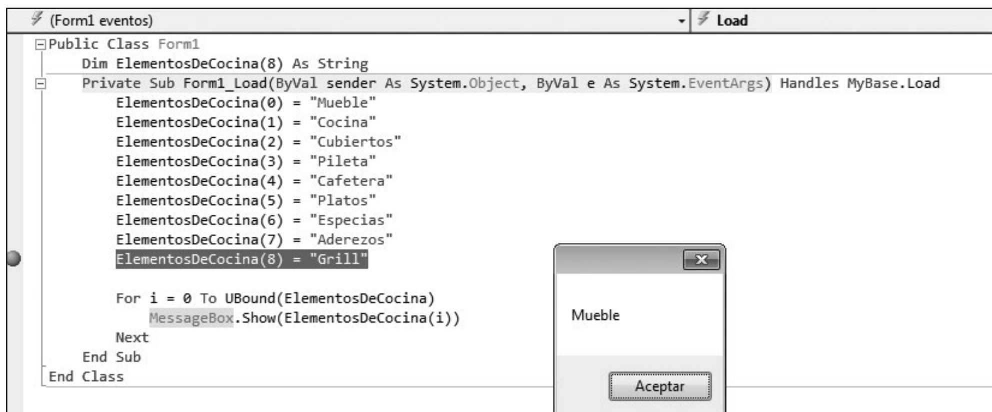


Figura 10. *Al finalizar la carga del array, se recorre toda la matriz de este y, a través de un cuadro de diálogo, se muestra cada valor cargado en ella.*

Cada array que declaremos puede redimensionarse en el momento en que sea necesario. Agregamos el siguiente código en el espacio en blanco que dejamos al escribir. De esta manera, añadimos una posición más al **array** sin perder el contenido ya cargado:

```
…
ElementosDeCocina(8) = "Grill"

ReDim Preserve ElementosDeCocina(9)
ElementosDeCocina(9) = "Ollas"

For i = 0 To UBound(ElementosDeCocina)
…
```

Veremos que, utilizando **ReDim Preserve**, podemos agregar una posición más al array sin perder lo que ya estaba cargado. Si ejecutamos el código, notaremos que los cuadros de mensajes ahora incluirán el nuevo **String**,**"Ollas"**.

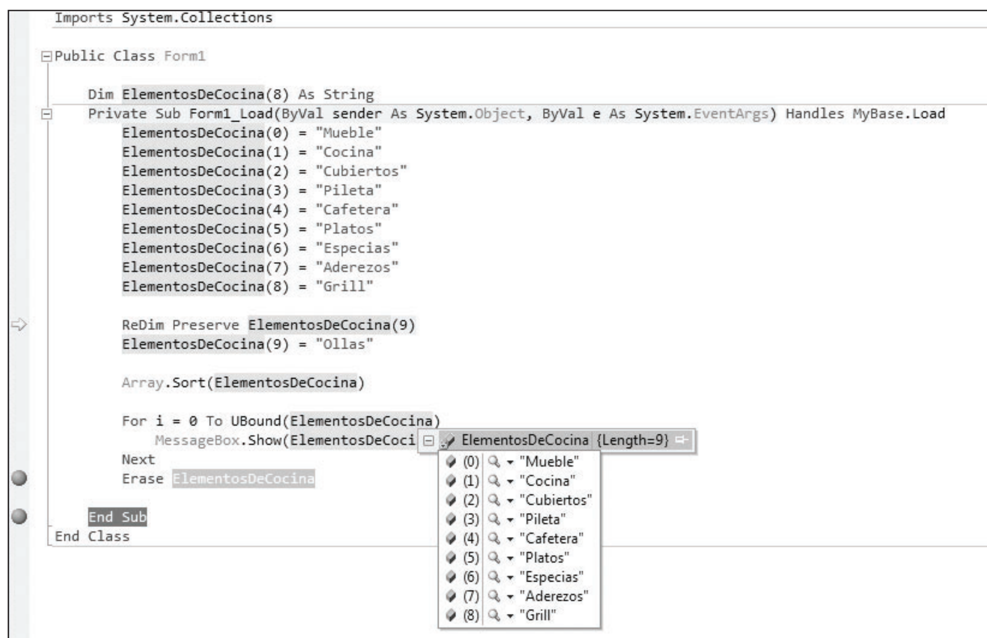


Figura 11. *Visual Basic 2010 tiene un inspector de elementos que nos permite ver el array completo de nuestra variable. Posicionando el cursor sobre ella, podemos desplegar todo su contenido.*

Si deseamos ordenar un array, debemos utilizar el método **Sort** contenido dentro de la colección **Array**. El orden que podemos darle es variado. En el ejemplo que

venimos utilizando, vamos a ordenar los datos cargados de manera alfabética. En el mismo código que teníamos, agregamos la siguiente instrucción:

```
…
Array.Sort(ElementosDeCocina)

For i = 0 To UBound(ElementosDeCocina)
…
```

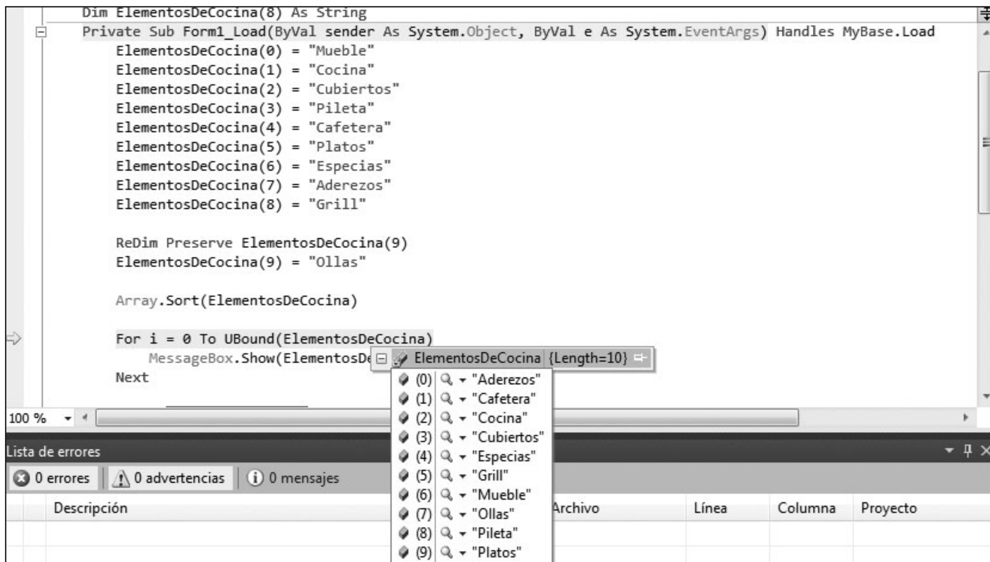


***Figura 12.** Gracias a Sort, el orden de nuestro array se hace en forma automática. Comparemos su contenido con la imagen anterior.*

Como todo elemento que creamos en el proceso de nuestro programa, el **array** ocupa espacio en memoria asignado al dimensionar la variable. Si no lo necesitamos, podemos eliminarlo una vez que se haya utilizado. Esto se realiza con la sentencia **Erase**, como indica el código a continuación:



PUNTOS DE INTERRUPCIÓN

Los puntos de interrupción dentro de los programas son ideales para analizar con detenimiento el comportamiento que va teniendo nuestra aplicación al momento de ejecutarse. Podremos ver el valor de las variables y las operaciones cuando se realizan. Con ellos aprenderemos a detectar errores que, a simple vista, no notamos.

```
Erase ElementosDeCocina
```

Las **enumeraciones** se agrupan dentro de una **estructura** que permite seleccionar cualquiera de los valores declarados en ella. Pueden ser declaradas solo con el tipo de dato entero; no pueden contener un valor decimal.

```
Public Class Form1
  Enum Hora As Short
    Cero = 0
    Uno = 1
    Dos = 2
    Tres = 3
    Cuatro = 4
    Cinco = 5
  End Enum
…
```

Si no especificamos un valor para cada componente de la estructura, este asumirá por defecto que contiene el valor de su índice, comenzando desde cero. Ahora, si no especificamos un valor para cada componente aunque estos tengan su nombre asignado a una lógica, y en algún momento de nuestra aplicación cambiamos el orden de alguno, dicha lógica se perderá. Hay que evaluar detenidamente los valores que se almacenarán los componentes para asignarlos o no, si es necesario, al momento de crear el **enumerador**.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
   MessageBox.Show("El primer valor del Enumerador Hora es: " & Hora.Cero)
End Sub
```

Colecciones

Las colecciones son clases que tienen un conjunto de valores asociados en el momento de su creación, como si se tratase de un **array**, con la diferencia de que podremos obtener el valor de cada elemento que compone la colección a través de su clave o de su índice. La versión actual de Visual Basic .NET mejoró la inicialización de una colección respecto a las ediciones anteriores. La reducción de código es significativa, y se puede realizar la llamada a un método de manera

repetitiva en una única instrucción. También es posible que Visual Basic detecte automáticamente el tipo de colección que estamos creando, con lo cual podemos o no indicarle qué tipo de dato contiene:

```
'Creación de una nueva colección de tipo String
Dim RevistasDeTecnologia As New List(Of String)({"USers", "Power Users",
"Dr. Max", "ExpandIT"})

'O también es válido de esta manera, sin especificar el tipo String
Dim librosDeProgramacion {"Visual Basic", "Visual C#", "Visual C++",
"Visual F#"}
```

En ambos casos, declaramos una colección de tipo **String**. En la primera declaramos el **tipo de lista** que íbamos a crear, mientras que en la segunda no lo hicimos. En las versiones previas a Visual Studio, este proceso se llevaba a cabo de la manera que vemos en las siguientes líneas de código:

```
    Dim revistasVarias As New List(Of String)
    revistasVarias.Add("Patoruzito")
    revistasVarias.Add("Satiricón")
    revistasVarias.Add("Barcelona")
```

Iniciamos un nuevo proyecto **Windows Form** para trabajar con las colecciones y comprender mejor cómo manejarse con ellas. Toda la declaración de las colecciones creadas se agrega en el inicio de la clase **Form1**:

```
Public Class Form1
  Dim RevistasDeTecnologia As New List(Of String)({"Users", "Power Users",
"Dr. Max", "ExpandIT"})
```



QUÉ OPCIÓN UTILIZAR

Por una cuestión de retrocompatibilidad en la migración de proyectos, como por el hecho de permitir que los programadores se acostumbren al cambio, ambas maneras son válidas para crear una colección, de modo que se mantienen en Visual Basic 2010. Igualmente, es mejor acostumbrarse al nuevo estilo, dado que en un futuro este soporte desaparecerá.

```
   Dim librosDeProgramacion As New List(Of String)({"Visual Basic", "Visual
C#", "Visual C++", "Visual F#"})
   Dim revistasVarias As New List(Of String)
…
```

Añadimos a continuación en **Form1** tres controles **ListBox**, y en el evento **Form_Load** escribimos el siguiente código:

```
For Each item In RevistasDeTecnologia
  ListBox1.Items.Add(item)
Next

For Each item In librosDeProgramacion
  ListBox2.Items.Add(item)
Next

revistasVarias.Add("Patoruzito")
revistasVarias.Add("Satiricón")
revistasVarias.Add("Barcelona")

For Each item In revistasVarias
  ListBox3.Items.Add((item))
Next
```

En el código escrito en el evento **Load**, utilizamos la sentencia **For Each** para recorrer cada uno de los elementos. Cuando se posiciona en un elemento, este queda asignado a **Item**, para luego ser agregado dentro de los ítems que contendrá **ListBox**.

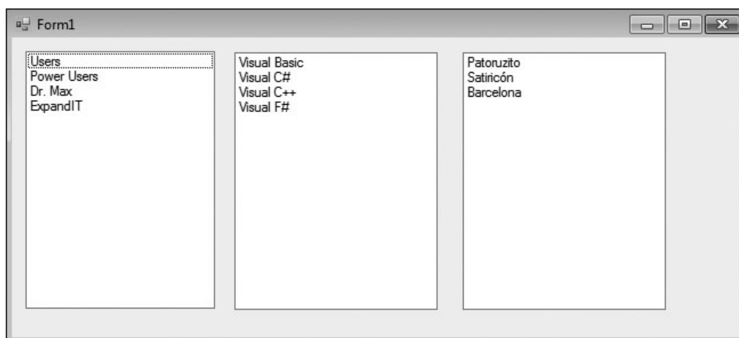


***Figura 13.** Todo el contenido de las tres colecciones creadas fue cargado a cada **ListBox** a través de la sentencia **For**.*

Las colecciones disponen de **métodos específicos** que permiten trabajar sobre su contenido; los más comunes son **Add**, **Clear**, **Count** y **Remove**. Vamos a modificar nuestro proyecto para incorporar ejemplos de dichos métodos. Eliminamos **Listbox2** y **Listbox3**, agregamos seis controles **Button** y establecemos la propiedad **Text** de cada uno de ellos con las acciones que realizarán: **"Limpiar"**, **"Eliminar"**, **"Agregar"**, **"Contar"**, **"Listar"** y **"Ordenar"**. Por último, escribimos el siguiente código:

```
Public Class Form1
  Dim librosDeProgramacion As New List(Of String) _
    ({
     "Visual Basic", "Visual C#", "Visual C++", "Visual F#", "Java", "PHP"
    })

  Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    librosDeProgramacion.Clear()
  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim i As Long = InputBox("Indique el índice a eliminar de un máximo de
 " & librosDeProgramacion.Count - 1, 0)
    If i > librosDeProgramacion.Count - 1 Then Exit Sub
    librosDeProgramacion.Remove(i)
  End Sub
```



USO CORRECTO DE FOR EACH

En Visual Basic .NET, la sentencia **For Each** permite ahorrar muchas líneas de código. Antes, había que obtener en una variable el valor máximo de la colección, y combinar con la sentencia **For** que recorriera desde el primer ítem hasta el último, para que fueran agregados en un control de listas. Esto demandaba no menos de siete líneas de código.

```
  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    Dim sNuevo As String = InputBox("Ingrese el nuevo libro a agregar en
la colección:")
    If sNuevo.Trim() <> "" Then librosDeProgramacion.Add(sNuevo)
  End Sub

  Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
    ListBox1.Items.Clear()
    For Each item In librosDeProgramacion
      ListBox1.Items.Add(item)
    Next
  End Sub

  Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    Dim i As Long = librosDeProgramacion.Count    MsgBox("Cantidad de
ítems en la colección: " & i)
  End Sub

  Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
    librosDeProgramacion.Sort()
    Button5_Click(sender, e)
  End Sub
End Class
```

Ejecutamos el proyecto presionando **F5** y probamos toda su funcionalidad. Ya logramos explorar y comprender el uso de todas las características que nos provee el uso de colecciones.



MATRICES MULTIDIMENSIONALES Y ESCALONADAS

Visual Basic permite manipular datos múltiples a través de matrices multidimensionales y escalonadas. Estas dan la posibilidad de expandir mucho más la funcionalidad vista en las colecciones, dado que ofrecen un control total de la información que necesitamos manipular. Es posible consultar MSDN en **http://msdn.microsoft.com/es-es/library/5dk93f6e(v=VS.80).aspx**.

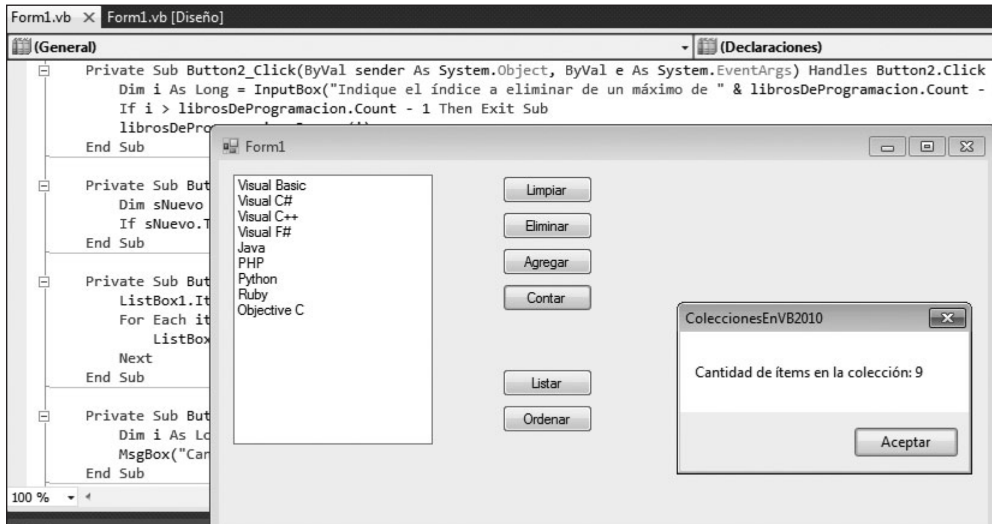


Figura 14. *Nuestro proyecto de manejo de colecciones ya es funcional.*

FORMULARIOS

En todo lenguaje de programación visual, ya sea de aplicaciones que corren sobre el entorno gráfico del sistema operativo como de aquellas que lo hacen dentro de un navegador, contamos con un área de trabajo en la cual distribuimos los controles y mostramos la información al usuario.

Qué es un formulario

Esta área de trabajo se denomina **formulario** (*Form*). Los formularios son objetos del tipo contenedor, ya que en ellos se distribuye toda la interfaz de nuestras aplicaciones, las cuales interactuarán mediante eventos o entrada de datos con los usuarios que las utilicen. Hasta ahora, vinimos trabajando de manera básica con un formulario. A continuación, veremos cuánto más se puede expandir su funcionalidad.

El formulario es un objeto que oficia de ventana contenedora dentro de Windows. Dispone de propiedades, eventos y métodos que permiten potenciar su uso al máximo. Veamos un detalle de los elementos principales de un **Form** a través de la **Tabla 5**.

PROPIEDAD	FUNCIÓN
Name	Especifica el nombre del formulario.
Text	Especifica el título de la ventana.
BackColor	Especifica el color del formulario.
ControlBox	Especifica si el formulario tiene un cuadro de menú.

PROPIEDAD	FUNCIÓN
Enabled	Especifica si el formulario habilita la función de sus controles contenidos o no.
Font	Especifica la fuente predefinida del formulario y todos los controles que este contenga.
Icon	Especifica un icono distintivo.
Location	Especifica las coordenadas X e Y en pantalla donde aparecerá.
MaximizeBox	Especifica si tiene un botón para maximizarlo.
MinimizeBox	Especifica si tiene un botón para minimizarlo.
MaximunSize	Especifica el tamaño máximo (ancho y largo) que tendrá.
MinimunSize	Especifica el tamaño mínimo (ancho y largo) que tendrá.
Opacity	Especifica su grado de opacidad.
ShowInTaskBar	Especifica si se mostrará en la barra de tareas de Windows.
WindowState	Especifica si se inicia minimizado, normal o maximizado.

Tabla 5. *Propiedades más comunes utilizadas en un formulario.*

Vamos a iniciar un nuevo ejemplo para trabajar con formularios y ver sus principales características. El proyecto a desarrollar a continuación será de tipo **Windows Form** y lo llamaremos **TrabajoConForms**.

Agregar más de un formulario

Agregamos en el formulario un botón y, dentro de él, escribimos el siguiente código:

```
Dim formu As New Form
formu.FormBorderStyle = FormBorderStyle.None
formu.SetDesktopLocation(500, 500)
formu.Show()
```

En este código creamos un nuevo objeto denominado **formu** como nuevo **Form**. Le indicamos que no tenga un borde específico. Lo ubicamos en las coordenadas **500**, **500** de la pantalla y lo mostramos. Otros valores aplicables a **FormBorderStyle** son: **Borde3D** y **FixedToolWindow**.
Los métodos más comunes utilizados con los formularios son:

- **Show()**: permite mostrar el formulario.
- **Hide()**: permite ocular el formulario.
- **ShowDialog()**: muestra el formulario en modo exclusivo, sin poder volver al anterior hasta que no cerremos este.
- **Close()**: cierra el formulario.
- **Focus()**: le da el foco principal a dicho formulario.

Iniciar y ocultar formularios

A nuestro proyecto **TrabajoConForms** incorporaremos un **Form** prediseñado para agilizar la tarea de desarrollo. Para hacerlo, vamos al menú **Proyecto/Agregar nuevo elemento** y, del cuadro de diálogo que se presenta, seleccionamos **Cuadro Acerca de**. Reemplazamos el código escrito en **Button1** por el siguiente:

```
AboutBox1.ShowDialog()
```

Iniciamos otra vez nuestra aplicación con **F5** y presionamos el botón. Veremos el cuadro de diálogo. La información mostrada por este **Form** será leída de los **datos del ensamblado**, que se especifican en el panel **Aplicación** del **Diseñador de proyectos**. Para cerrar el diálogo **Acerca De**, solo debemos hacer clic en el botón **Aceptar**, que contiene el código para cerrar dicho **Form**.

```
Me.Close()
```

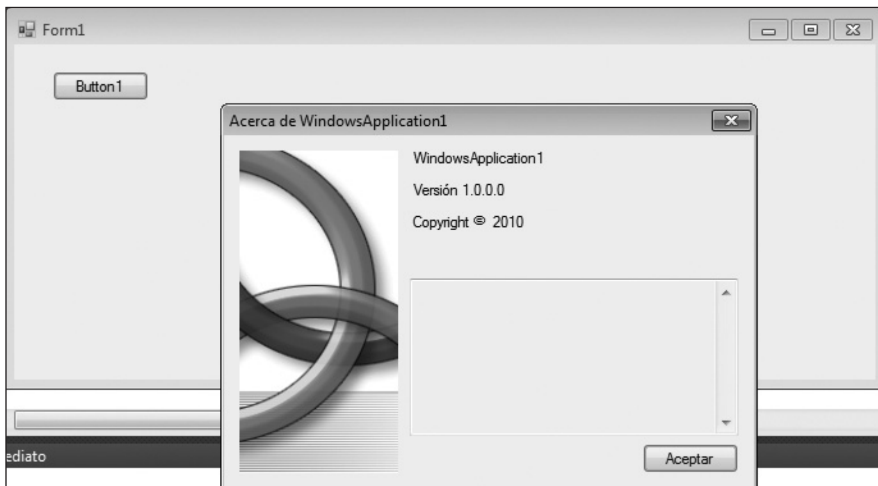


***Figura 15.** El formulario Acerca De creado desde elementos prediseñados que contiene Visual Basic.*

INSTRUCCIONES BÁSICAS

Todo lenguaje de programación posee **instrucciones básicas**, que son las necesarias para que el programa cumpla su ciclo de procesamiento de datos y entrega de resultados. Al principio de este capítulo hablamos de **algoritmos**, y dijimos que está

compuesto por estas órdenes que permiten llevar a cabo una o más tareas. A continuación, repasaremos las principales instrucciones básicas de Visual Basic.

If else

La instrucción **if** permite determinar si una condición se cumple o no, y sobre esa base, ejecutar otra orden; **if** siempre va acompañada por **Then**, que es la que indica la acción a ejecutar según.
A su vez, podemos agregar un segundo actor, denominado **else**, que permite establecer qué ocurre si lo anterior no cumple la condición. Con **else** podremos ejecutar otra orden si es que la primera no se ejecutó.

Estructura If else

```
'Estructura if
If condición es verdadera then
  Hacer algo
Endif

'Estructura if else
If condición es verdadera then
  Hacer algo
else
  Hacer otra cosa
endif
```

Ejemplo If else

```
'Estructura if
Dim i as integer = 100
If i = 100 then
  i = 99
Endif

'Estructura if else
Dim i as integer = 100
If i = 1 then
  i = i + 1
else
```

```
  i = 1
endif
```

Elseif

Se denomina **Elseif** a una condición que se anida dentro de if, con el fin tener más de una posible alternativa para procesar otro código.

Estructura Elseif

```
If condición = algo then
Hacer esto
Elseif condición = otracosa then
Hacer esto otro
Elseif condición = algunaotracosa then
Haz esto otro
Else
No hagas nade de lo anterior, haz esto
End if
```

Ejemplo Elseif

```
If Variable = 1 then
Variable = 2
Elseif Variable = 2 then
Variable = Variable * 2
Elseif Variable = 4 then
Variable = Variable / 2
Else
Variable = 0
End if
```

For Next

La sentencia **For** crea un bucle que va desde **un valor mínimo** hasta el **máximo especificado**, ejecutando el código que tenga en su interior.
Asimismo For puede contener otras instrucciones condicionales en su interior.

```
Estructura For Next
For K = valorInicial to valorFinal
Letra = Letra + " a "
Next
```

Para ejecutar correctamente la instrucción **For**, debemos especificar un valor inicial y un valor final a la sentencia.

Ejemplo For Next

```
Dim L as long = 0
Dim K as long
For K = 1 to 20
L = L + 15
Next i
```

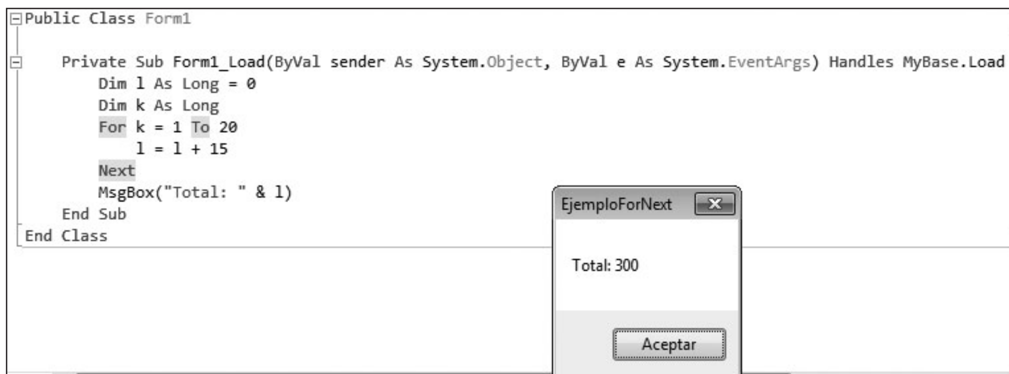


***Figura 16.** For recorrió 30 veces su ciclo sumando 15 unidades cada vez.*

Do while loop

Do while es una estructura repetitiva que permite crear un bucle que se ejecutará mientras una condición sea cierta o no cambie su valor.

Estructura Do While Loop

```
Do while estaCondicionSeaCierta
Código…
Loop
```

Ejemplo Do While Loop

```
'Dim Hora as integer = 0
Do while Hora < 13
Hora += 1
Loop
```

For Each

Hemos utilizado **For Each** cuando trabajamos con colecciones. Esta sentencia crea un **bucle** que interactúa con determinados valores y devuelve una referencia a ellos.

Estructura For Each

```
Por cada DLL que encuentres en Windows\system32
cargala en listBox1
Siguiente
```

Ejemplo For Each

Vamos a crear una aplicación **Windows Form** con un **listBox** y un botón. Dentro de **Button1** escribimos el siguiente código:

```
    ListBox1.Items.Clear()
    For Each Archivo As String In My.Computer.FileSystem.GetFiles( _
          "c:\windows\system32", _
FileIO.SearchOption.SearchTopLevelOnly, _
                "*.dll")
      ListBox1.Items.Add(Archivo)
    Next
    Label1.Text = ListBox1.Items.Count
```



AMPLIAR FUNCIONES DE SENTENCIAS

Cada una de las sentencias del lenguaje posee infinitas características y variantes de uso. Incluso, para llegar a una solución muchas veces nos encontraremos en la duda de si usar una sentencia u otra. El resultado exacto de esto solo lo tiene la experiencia de los años, que es la que nos asegurará cuál es la correcta.

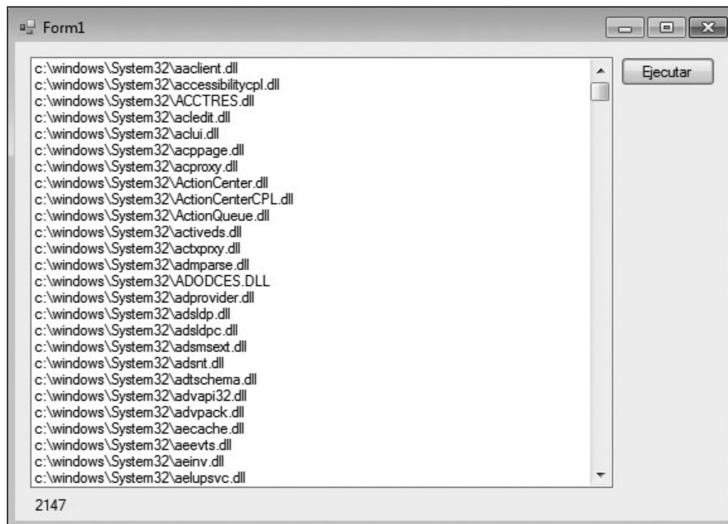


Figura 17. *Con el uso de **For Each**, y en solo cuatro líneas de código, podemos listar los archivos contenidos en una carpeta.*

Select case

Esta sentencia se asimila a **If Else**, aunque le agrega la posibilidad de evaluar más de una condición al mismo tiempo y ejecutar una acción sobre cada una de ellas.

Estructura Select case

```
Select case {expresión}
Case {estevalor}
   …haz esto
Case {otrovalor}
   …haz lo otro
Case else
   …cambia esto
End select
```

Ejemplo Select case

```
Select case Hijo
Case “Nicolas”
   Alumno.edad = 14
Case “Julián”
```

```
    Alumno.edad = 5
Case else
    Alumno.edad = 0
End select
```

Procedimientos

Los **procedimientos** se componen de una **serie de instrucciones** o algoritmos que se ejecutan de manera consecutiva con un principio y un fin. Cada objeto que compone nuestro proyecto contiene procedimientos internos vinculados a sus eventos para que realicen determinadas acciones. Los procedimientos nos facilitan también dividir nuestro programa en porciones pequeñas de código que nos permiten comprender mejor qué es lo que hacen. A su vez, son reutilizables; es decir, podemos crear un procedimiento y usarlo en distintas partes de un programa sin tener que repetir el código. Un procedimiento puede ser declarado de distintas maneras: **Public**, **Private** y **Friend**.

```
Public Sub NombreDelProcedimiento (parámetros)
     Código a procesar…
End Sub
```

Hasta ahora hemos venido escribiendo código en los ejemplos prácticos llevados a cabo tanto en el evento **Form_Load()** como en eventos **Button_Click()**. Para los respectivos objetos (**Form** y **Button**), estos son sus eventos, que en realidad están conformados por un procedimiento.

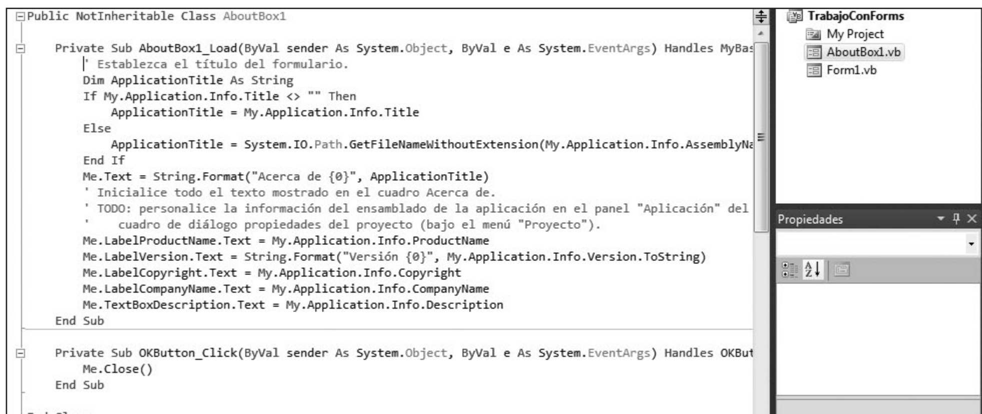


Figura 18. *El evento **Load** de **AboutBox1** obtiene, a través de un procedimiento, la **Versión del ejecutable**, el **Copyright** y el **Nombre de la aplicación**.*

Crear un procedimiento

Además de los procedimientos básicos creados por Visual Basic y vinculados a cada evento de nuestro programa, también podemos crear nuestros propios procedimientos. Iniciemos un nuevo proyecto **Windows Form** llamado **CrearProcedimientos** para comprender mejor cómo trabajar con ellos.
En **Form1** agregamos dos botones. En la propiedad **Button1.Text** escribimos **"Muestro mensaje"**; y en la propiedad **Button2.Text** escribimos **"Vuelvo a mostrarlo"**. A continuación, escribimos el código siguiente para que los eventos de este proyecto se vuelvan funcionales:

```
Public Class Form1
  Public Sub MuestroMensaje(msg as string)
    Dim msg As String = "Este mensaje se visualiza desde " & Me.Name
    MessageBox.Show(msg)
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    MuestroMensaje()
  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    MuestroMensaje()
  End Sub
End Class
```

Nuestro código muestra que, luego de la inicialización de la clase **Form1** creamos un procedimiento de tipo **Public** llamado **MuestroMensaje**. Este procedimiento tiene un **cuadro de diálogo** que muestra un mensaje. En el evento **Click** de cada **Button** llamamos al procedimiento para que se ejecute y nos muestre el cuadro de diálogo. Ejecutemos el proyecto para ver si funciona.
Los procedimientos pueden recibir uno o más parámetros adicionales que complementen su acción. Modifiquemos **MuestroMensaje** para que reciba un parámetro, así su acción cambiará dependiendo de un valor:

```
  Public Sub MuestroMensaje(ByVal msg As String)
    MessageBox.Show(msg)
  End Sub
```

Aquí le indicamos un parámetro adicional del tipo **String**. A través de él se recibirá el mensaje que el cuadro de diálogo debe mostrar. A continuación, escribamos el código correspondiente para cada control **Button**:

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    MuestroMensaje("Mensaje mostrado desde button1")
  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    MuestroMensaje("El mismo procedimiento llamado desde button2")
  End Sub
Al ejecutar el proyecto, veremos el resultado.
```

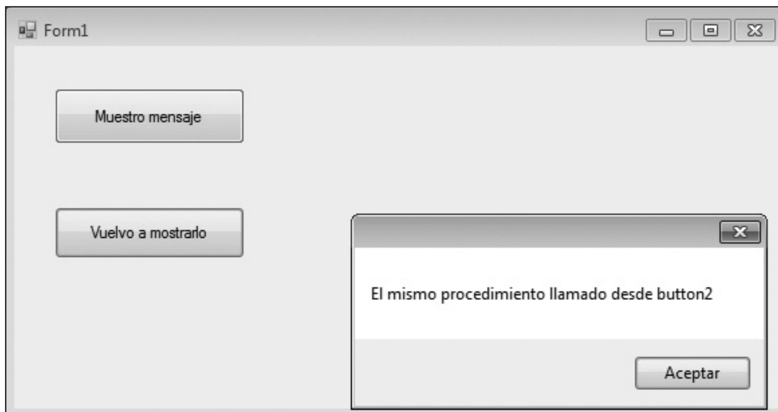


Figura 19. *Con el mismo procedimiento que ahora recibe un parámetro podemos llegar al mismo resultado.*

Funciones

Las **funciones** son iguales a los procedimientos, con la diferencia de que ejecutan su código interno y deben devolver un resultado. Dicho resultado puede ser de cualquier tipo de dato: **String**, **Boolean**, **Integer**, **Double**, y otros. Al igual que los procedimientos, las funciones pueden recibir parámetros adicionales o no.

```
'Función sin parámetros
Public Function obtengoUltimoNumero() as long
…proceso
```

```
obtengoUltimoNumero = resultadoObtenido
End Function

'Función con parámetros
Public Function RealizoOperacion(NroA as single, NroB as single) as Double
RealizoOperacion = NroA * NroB
End Function
```

Funciones sin parámetros

Para comenzar, iniciamos un nuevo proyecto **Windows Form** llamado **TrabajoConFunciones** y agregamos dos **Forms** más, adicionales a **Form1**. En **Form1** añadimos dos controles **Button**: uno con la propiedad **text "Funciones sin parámetros"** y el otro con la propiedad **text "Funciones con parámetros"**.

En **Form2** agregamos un control **TextBox** y uno **Button**. A continuación del inicio de la clase **Form2**, creamos el siguiente procedimiento:

```
'A través del siguiente código, podemos ver de qué manera utilizar
'la condición IF, dependiendo de distintas variables.
  Public Function averiguoTipoDeDato() As String
    Dim valor As String
    valor = Trim(TextBox1.Text)
    If valor = "" Then
      averiguoTipoDeDato = "El TextBox no contiene información."
      Exit Function
    End If
    If valor <> "" Then
      averiguoTipoDeDato = "El TextBox contiene esto: " & valor
      Exit Function
    End If
    averiguoTipoDeDato = "No fue procesada ninguna de las opciones."
```



OPERADORES DE COMPARACIÓN

Los operadores de comparación suelen utilizarse dentro de operaciones con números como para comparar textos o diversos estados de objetos y controles. Ellos son: **=** (igual que), **<** (menor que), **>** (mayor que), **<=** (menor o igual que), **=>** (mayor o igual que) y **<>** (distinto que).

```
End Function
```

En **Button1** agregamos el siguiente código:

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    MsgBox(HayValorIngresado)

  End Sub
```

Button1 ejecuta un cuadro de diálogo simple que recibe como parámetro el texto devuelto por la función **HayValorIngresado**.

Funciones con parámetros

Las funciones con parámetros trabajan de la misma manera que los procedimientos que reciben parámetros, y también devuelven un resultado, que siempre está atado a los valores que les pasamos en su ejecución. Para comprender mejor este tema, en **Form3** vamos a agregar dos **TextBox** y un **Button**. En el inicio de la clase **Form3** creamos la siguiente función:

```
  Public Function TotalDeSuma(ByVal tUno As Single, ByVal tDos As Single)
As Double
    If IsNumeric(tUno) And IsNumeric(tDos) Then
      TotalDeSuma = tUno + tDos
      Exit Function
    End If
    TotalDeSuma = 0
```

Por último, en **Button1** escribimos el siguiente código:



ESTRUCTURAS Y OPERADORES

La mayoría de las estructuras condicionales siempre van acompañadas por un operador para saber distinguir si tienen que ejecutarse o no. En los casos de las funciones que devuelven una expresión booleana como resultado, los operadores de comparación pueden obviarse.

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    MsgBox(TotalDeSuma(TextBox1.Text, TextBox2.Text))
  End Sub
```

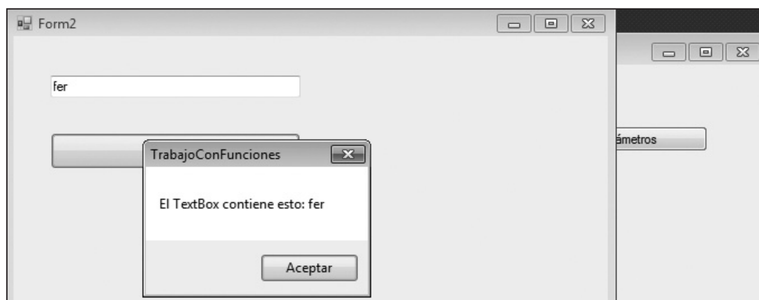


***Figura 20.** Las funciones, al igual que los procedimientos, nos permitirán tener procesos a mano sin necesidad de repetir código.*

Funciones propias del lenguaje

Visual Basic dispone de funciones internas, algunas integradas en cada espacio de nombres, y otras como funciones miembro que se utilizan dentro de la ejecución de una aplicación. En ellas se agrupan diversas funciones, como las siguientes:

- **Funciones de conversión**: permiten convertir un tipo de objeto en otro:

```
'cBool convierte en True o False una expresión definida. Ésta debe ser del
tipo numérico y sólo 1 ó 0.
cBool(1) 'Convierte en True dicho valor
cBool(0) 'Convierte en False dicho valor

'cChar permite transformar en caracteres de texto cualquier valor que se le
 pase.
```



DECLARACIÓN DE FUNCIONES

Si una misma función debe trabajar en varios **Forms** de nuestro programa, se deberá declarar en un **módulo** en vez de hacerlo al inicio de una clase Form. Además, indefectiblemente debemos especificarle parámetros si debe realizar cálculos. Esto nos garantiza disponer de ella en cualquier rincón de nuestro proyecto.

```
cChar(True) 'Devolverá "True" como palabra
cChar(123.45) 'Devolverá "123.45" como texto
```

- **Funciones matemáticas**: permiten realizar conversiones u operaciones con números de cualquier tipo de dato:

```
'Abs devuelve el valor absoluto de un número del tipo Double
Abs(1234.563) 'Devuelve 1234

'Rnd devuelve un valor Entero más cercano al punto de redondeo
Rnd(444.40) 'Devuelve 444.0
Rnd(444.61) 'Devuelve 445.0
```

En el ejemplo anterior, hacemos uso de dos funciones: **Abs()**, quien devuelve el valor absoluto de un número, y **Rnd()**, que ejecuta un redondeo hacia el número anterior más próximo.

- **Otras funciones**: son funciones que permiten transformar un tipo a otro, con ciertas limitaciones entre el **tipo de origen** y **tipo de destino**. Incluso, pueden generar mensajes o ventanas adicionales:

```
'Los cuadros de diálogos son funciones propias del sistema.
Messagebox.show("Este es un cuadro de diálogo.")

'Verificar si una variable es numérica.
Dim numero as Object = 123
Dim a as boolean = isNumeric(numero) 'Devuelve True

'Transformación de textos.
Dim a as string = "cadena de texto"
```



CARGA DE DATOS

En aquellos casos en que hay que cargar datos de una matriz, colección o variable con array, lo mejor para hacerlo es utilizar **For Each** en vez de **For Next**, debido a que el primero fue optimizado para que realice su proceso de manera más rápida y sin tener que efectuar pasos previos, como la obtención del número final de elementos por procesar.

```
Dim b as string = a.ToUpper 'Devuelve "CADENA DE TEXTO"
Dim c as string = "  Otra cadena de caracteres  "
Dim d as string = c.Trim 'Devuelve "Otra cadena de caracteres"
Dim e as string = c.TrimStart 'Devuelve "Otra cadena de caracteres  "
Dim f as string = c.TrimEnd 'Devuelve "Otra cadena de caracteres"


'Formateo de fechas
Dim dia as date = "15/11/2010"
Dim a as string = dia.ToString 'Devuelve "15/11/2010"
Dim b as string = format(dia,"DD-MM-YY") 'Devuelve "15-11-10"
Dim dia As Date = "15/10/2010"
Dim fecha As String = dia.ToLongDateString 'Devuelve "Viernes 15 de Octubre
 de 2010"
```

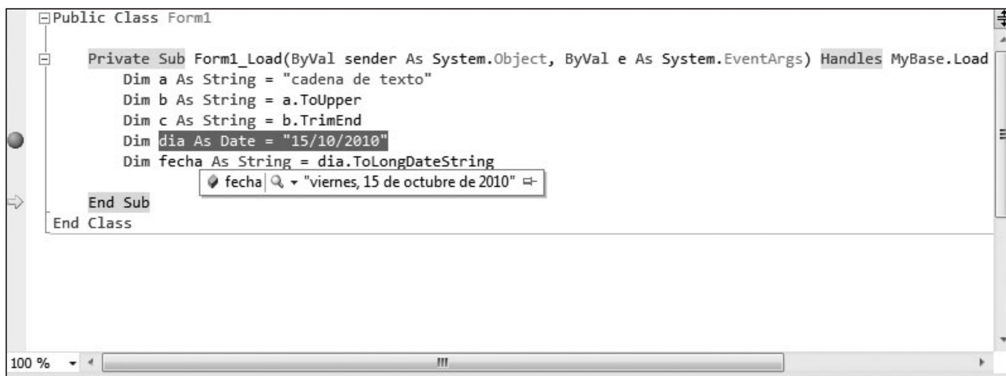


***Figura 21.** Podemos observar, en la pantalla de Visual Basic 2010, que la variable ya obtuvo la fecha en formato largo, gracias a la función integrada **ToLongDateString**, explicada anteriormente.*

EJEMPLO PRÁCTICO CON PROCEDIMIENTOS Y FUNCIONES

Iniciemos un nuevo proyecto para integrar alguno de los conceptos aprendidos hasta este momento. Para esto, seleccionamos como tipo de proyecto **Windows Form** al cual llamaremos **EjemploPracticoUno**.
En **Form1** agregamos los siguientes controles:
- **MonthCalendar**

- **TextBox** sin texto inicial
- **CheckBox** con su valor desmarcado
- **Button** con el texto **"Go"**
- **ListBox**
- **Button** entre los **ListBox**: uno con texto **"Pasar a MAYUS"** y el otro con el texto **"Pasar a minusc"**.
- **Button** con el texto **"Cargar colección"** debajo de **ListBox1**.



***Figura 22.** Este es el diseño que debe tener nuestro **Form**.*

A continuación, entramos en el modo código de **Form1** y añadimos el código debajo listado en cada uno de los controles que agregamos:

```
Public Class Form1
  Dim Coleccion As New List(Of String)({"iSpeakers", "Parlantes 2.0",
"Parlantes 2.1", "Parlantes 5.1", "Parlantes 7.1"})
  Function chequeoValorIngresado(ByVal i As Object) As Boolean
    chequeoValorIngresado = False
```

{ } ÍNDICE DE LOS ARRAYS

Desde la versión 2005 , la cláusula **Option Base** dejó de incluirse cuando había que trabajar con un grupo de arrays. Por defecto, cualquier matriz de datos se inicializa desde cero en lugar de otro número. Esto fue así porque en la versión 2003 o ulterior, muchos arrays podían ser no iniciados desde cero, y esto causaba gran confusión entre programadores de otros lenguajes.

```
    If Not IsNumeric(i) Then Exit Function
    If i <> 0 And i <> 1 Then
      Exit Function
    Else
      chequeoValorIngresado = True
      Exit Function
    End If
  End Function

  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    'Cargar colección
    For Each item In Coleccion
      ListBox1.Items.Add(item)
    Next
  End Sub

  Private Sub MonthCalendar1_DateSelected(ByVal sender As Object, ByVal e
As System.Windows.Forms.DateRangeEventArgs) Handles MonthCalendar1.DateSe
lected
    'Día seleccionado del calendario
    Dim dia As Date = e.Start.Date
    TextBox1.Text = dia.ToLongDateString
  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    'Cambiar estado del checkBox1
    Dim j As Object = InputBox("Indique con 1 ó 0 si quiere o no tildar el
 CheckBox", "Marcar checkBox", 0)
    If chequeoValorIngresado(j) Then CheckBox1.Checked = CBool(j)
  End Sub

  Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    'Pasar de listBox1 a listBox2 a mayúsculas
    If ListBox1.SelectedIndex <> -1 Then
      ListBox2.Items.Add(ListBox1.Text.ToUpper)
    End If
  End Sub
```

```
  Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    'Pasar de listBox1 a listBox2 a minúsculas
    If ListBox1.SelectedIndex <> -1 Then
      ListBox2.Items.Add(ListBox1.Text.ToLower)
    End If
  End Sub
End Class
```

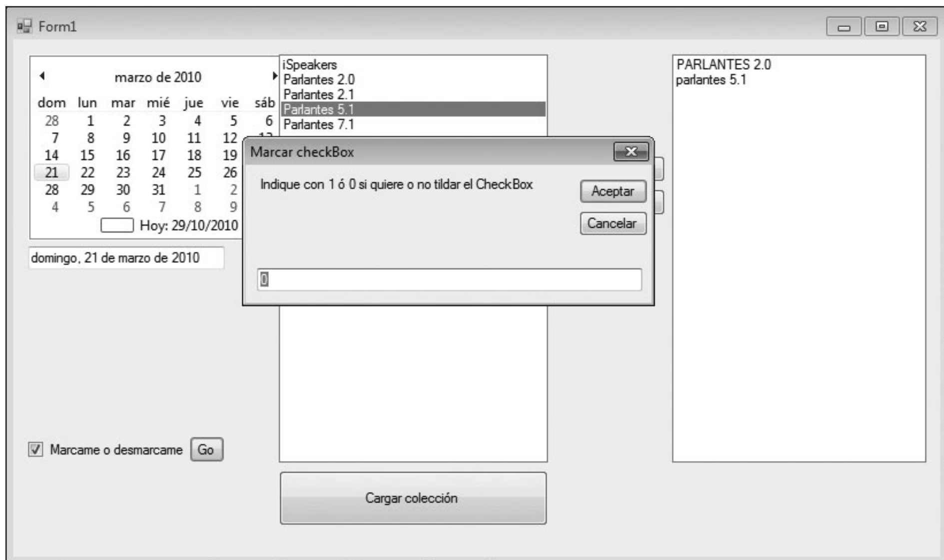


Figura 23. *Nuestro proyecto terminado y funcional. En él se aplicaron varios de los objetivos vistos en este capítulo.*

RESUMEN

Hasta aquí hemos visto un pantallazo de los fundamentos del lenguaje y cómo se estructura la mayoría de las aplicaciones por dentro. También repasamos las instrucciones principales del lenguaje, las cuales seguramente nos acompañarán desde aquí hasta el final del libro. Si bien Visual Basic 2010 contiene muchas más instrucciones, las principales que utilizaremos a menudo en los desarrollos son las estudiadas durante este capítulo. Las funciones y la creación de procedimientos personalizados también será algo que utilizaremos todo el tiempo, lo que nos garantizará tener que repetir código en varias partes de una aplicación, y por consiguiente, poder corregirlo o mejorarlo en un solo lugar, y que esto quede aplicado a todo el proyecto de manera automática.



ACTIVIDADES

PREGUNTAS TEÓRICAS

1. Sobre la última aplicación creada, agregue botones que permitan ordenar la colección de manera alfabética y luego volver a cargarla en el **ListBox**.

2. ¿Los procedimientos que reciben parámetros devuelven algún valor como resultado?

3. ¿Las colecciones unidimensionales tienen un límite máximo de valores?

4. Investigue qué son las Jagged Collections y en qué difieren de las MultiDimensional Collections.

5. ¿Existe la función CInt()? ¿Y la función **CLong()**? Analícelas.

6. ¿Visual Basic 2010 permite crear matrices multidimensionales desde dos matrices unidimensionales?

7. ¿Cuál es el límite inferior que puede tener un array o colección? ¿Es posible cambiarlo por el límite menor que se quiera?

8. ¿Qué tipo de procedimiento debe utilizarse para que retorne un valor con un resultado?

9. ¿Es posible cambiar el tipo de valor de una variable una vez creada?

10. Cree una variable del tipo **Date** con la fecha de hoy y transforme la misma a una variable del tipo **String** en todos sus formatos de fecha posibles.

Namespaces y controles

En el Capítulo 3 de Visual Basic estudiaremos el significado de los espacios de nombre en .NET, cómo se compone su árbol, los distintos namespaces y para qué está orientado cada uno de ellos. También haremos una introducción a los controles básicos con los cuales desarrollaremos la interfaz visual de nuestras aplicaciones. Por último, integraremos los conocimientos adquiridos en un ejemplo práctico.

NAMESPACES

La llegada del **framework .NET** trajo consigo una serie de cambios sobre la estructura de cualquier lenguaje de programación de la plataforma Windows: los **espacios de nombre** o **namespaces**, en inglés. Ellos se encargan de organizar de manera jerárquica todos los objetos que se definen dentro del entorno. Los ensamblados pueden contener varios espacios de nombre distintos, que, a su vez, pueden incluir otros tantos espacios de nombre. Durante los primeros capítulos hemos visto que **Visual Basic .NET** dispone de muchas funciones básicas que nos ayudan a resolver cuestiones de programación en tan solo una línea de código. Dichas funciones, que se mantienen de manera compatible con las versiones anteriores del lenguaje, fueron agrupadas en diferentes namespaces dentro del framework .NET. Como los namespaces son estructuras definidas dentro del framework .NET, quien en algún momento deba utilizar otro lenguaje distinto de Visual Basic .NET, podrá familiarizarse rápidamente con el nuevo entorno de desarrollo.

My namespace

Visual Studio 2010 pone a nuestra disposición características que nos facilitan el desarrollo de aplicaciones, con lo cual nos permiten mejorar nuestra productividad y tiempos de desarrollo. El namespace `My` se utiliza para acceder a la información separada por instancias de objetos que componen la librería de clases base.
`My` es el nivel más superior desde el cual parten todos los objetos de .NET. Cada uno de ellos se comporta de la misma manera que un espacio de nombre o una clase.

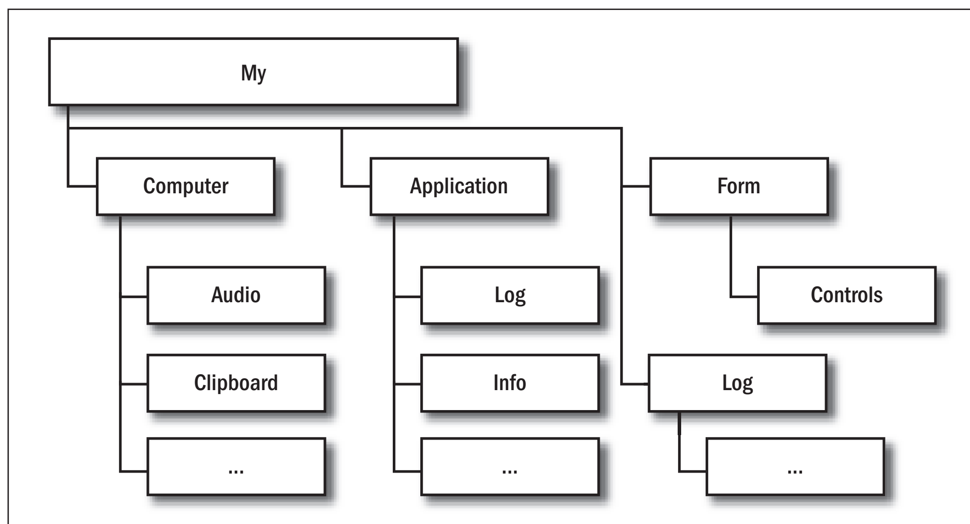


***Figura 1.** Ejemplo de la estructura principal de namespaces. La raíz se inicia con `My`, y **agrupa los recursos de acceso al software y hardware** desde el sistema operativo.*

Los objetos que se desprenden de **My** no siempre funcionan en todo el ámbito de desarrollo dentro de Visual Basic .NET. En la **Tabla 1** podemos observar en qué tipos de aplicaciones .NET funcionan y en cuáles no.

MY...	WINDOWS FORMS	BCL	CONSOLE APPS	BIBLIOTECA DE CONTROLES DE WINDOWS	BIBLIOTECA DE CONTROLES WEB	SERVICIOS DE WINDOWS	SITIO WEB
Application	X	X	X	X		X	
Computer	X	X	X	X	X	X	X
Forms	X			X			
Log							X
Request							X
Resources	X	X	X	X	X	X	
Response							X
Settings	X	X	X	X	X	X	
User	X	X	X	X	X	X	X
Webservices	X	X	X	X	X	X	

***Tabla 1.** Los objetos My marcados con una **X** están habilitados para ocho tipos de proyectos que pueden llevarse a cabo en **Visual Basic .NET**, como en **Visual Studio .NET**.*

Veamos un poco más en profundidad los objetos **My** que podemos utilizar en el desarrollo de aplicaciones **Windows Forms**.

My.Forms

My.Forms nos permite acceder a las instancias de cada formulario que componen nuestro proyecto **Windows Forms**. Cuando lo utilicemos, solo se expondrán las propiedades u objetos que se desprenden del objeto **Form** asociado a nuestro proyecto actual.

Por ejemplo, si quisiéramos conocer todo el contenido dentro de un **Form** (controles, funciones públicas, funciones propias del **Form** o sus controles, métodos, etc.) deberíamos utilizar **My.Forms.NombreDelForm**. De todas maneras, no es necesario utilizar toda la estructura completa para trabajar dentro de sí mismo. Para referenciar los controles o eventos propios de **Form1**, basta con denominarlo como **Me** seguido de la función correspondiente en vez de **My.Forms**…



MISMO NOMBRE PARA DOS FORMS

Si bien es difícil que en un proyecto coincidan dos formularios con el mismo nombre, esto no es imposible. Puede que un **Form** se encuentre en el raíz **WindowsApplication1**, y el otro, en el espacio de nombres **Namespace1**. Para acceder a este último, deberíamos hacerlo a través de **My.WindowsApplication1_Namespace_Form1**.

Iniciemos un nuevo proyecto **Windows Forms**, agreguemos algunos controles a **Form1** (entre ellos, un **Button**), y luego, en **Form_Load** escribamos lo siguiente:

```
My.Forms.Form1.Button1.Text = “Hola”
```

Veremos que el entorno de desarrollo marca un error subrayando algunas palabras con un trazo sinuoso de color azul. Esto es el indicador que debemos cambiar algo en nuestro código. En la **Figura 2** siguiente podemos ver un ejemplo de por qué no se puede lograr esto.

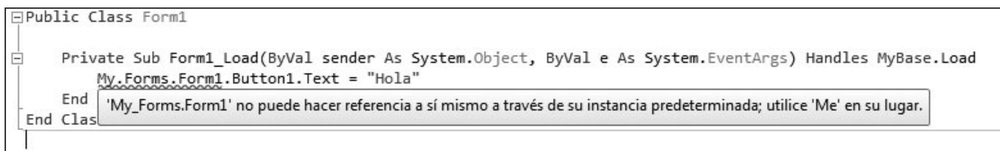


***Figura 2.** El código nos propone reemplazar My.Forms.Form1 por Me, para evitar que ocurra un error futuro si escribimos código referenciando a Form1 y, luego, cambiamos su nombre por otro.*

Para poder hacer uso de **My.Forms.Form1**… y referenciar o establecer el valor para alguno de sus componentes, esto debemos hacerlo desde otro **Form** o desde un **módulo de clase** o **módulo general**.

My.Computer

My.Computer nos permite acceder a todos los recursos componentes de nuestro equipo a través del sistema operativo. Dentro de esta clase, se desprenden más subclases, cada una de ellas hace referencia a algún recurso de hardware o software de la computadora.

MY.COMPUTER.	DESCRIPCIÓN
Audio	Permite acceder a los sonidos propios de Windows (no a MP3 o demás formatos de audio que necesitan un códec de por medio).
Clipboard	Permite acceder al contenido del Portapapeles.
Clock	Permite acceder al recurso de la hora.
Filesystem	Permite acceder a la estructura de archivos, carpetas y directorios, como también a las unidades de disco.
Info	Permite acceder a la información de los recursos de hardware.
Keyboard	Permite acceder al teclado.
Mouse	Permite acceder al mouse.
Name	Permite acceder al nombre de nuestra computadora.
Network	Permite acceder a los recursos de red.
Ports	Permite acceder a la información de los puertos.

MY.COMPUTER.	DESCRIPCIÓN
Registry	Permite acceder al Registro de Windows.
Screen	Permite acceder a las propiedades y métodos de la pantalla.

***Tabla 2.** My.Computer es un recurso utilizado por casi toda persona que desarrolle aplicaciones **Windows Form**.*

Para conocer mejor las prestaciones de **My.Computer**, vamos a iniciar un nuevo proyecto **Windows Forms**, en el que agregamos un **Listbox**, y **Labels** cuyas propiedades **Text** contengan los siguientes textos:

- **Nombre del equipo:**
- **Conectado a la red:**
- **Sistema operativo:**
- **Idioma instalado:**
- **Memoria física:**
- **Plataforma:**

Luego, en el evento **Form_Load** escribimos lo siguiente:

```
ListBox1.Items.Add(My.Computer.Name.ToString)
ListBox1.Items.Add(My.Computer.Network.IsAvailable())
ListBox1.Items.Add(My.Computer.Info.OSFullName.ToString)
ListBox1.Items.Add(My.Computer.Info.InstalledUICulture.ToString)
Dim a As Single = (My.Computer.Info.TotalPhysicalMemory / 1024)
ListBox1.Items.Add(Format(a, "###,###,###") + " Megabytes")
ListBox1.Items.Add(My.Computer.Info.OSPlatform.ToString)
```

My.computer.Name proporciona el nombre de la computadora. **My.Computer.Network.IsAvailable** indica si la máquina está o no conectada a la red. **My.Computer.Info.OSFullName** brinda el nombre completo del sistema operativo, en tanto que **My.Computer.Info.InstalledUICulture** nos dice qué idioma está instalado. **My.Computer.**



MY.APPLICATION.OPENFORMS

Dentro de la clase **Application** disponemos de **OpenForms**, que nos devolverá todos los formularios abiertos en nuestra aplicación en tiempo de ejecución. Cada form que vayamos a abrir dispone de una propiedad **InvokeRequired**, que nos permite saber el estado de cada formulario antes de acceder a él.

Info.TotalPhysicalMemory indica cuánta memoria tiene la computadora, y **My.Computer.Info.OSPlatform**, si el procesador es de 32 o 64 bits.

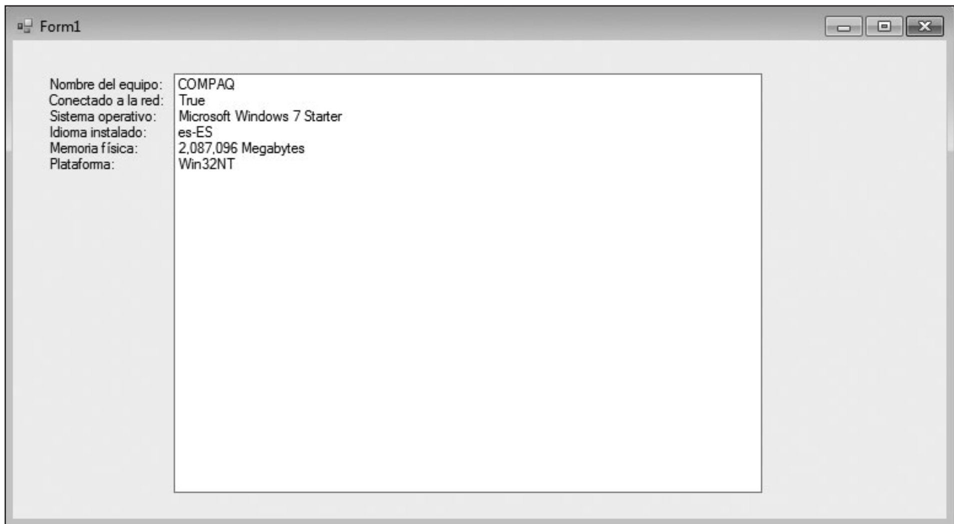


Figura 3. *El resultado de nuestra aplicación. Con pocas líneas de código, obtuvimos un informe de la computadora en la cual trabajamos con Visual Basic .NET 2010.*

My.Application

My Application agrupa todas las propiedades, métodos y eventos relacionados con la aplicación actual. A través de él, solo podemos tener acceso a las propiedades que exponen los datos agrupados en el objeto en sí, sin posibilidad de modificarlos. Esta información generalmente está disponible para **Windows Forms** o **aplicaciones de consola**. Algunos de los objetos que agrupa esta clase son:

MY.APPLICATION	DESCRIPCIÓN
ChangeCulture	Cambia la referencia cultural para manipular cadenas de caracteres en otro idioma.
ChangeUICulture	Cambia la referencia cultural de un subproceso en uso para permitir recuperar recursos específicos.
Culture	Obtiene la referencia cultural que se está utilizando en el subproceso actual.
Deployment	Permite implementar ClickOnce para actualizar la aplicación en uso y descargar archivos a petición.
DoEvents	Permite procesar mensajes de Windows en un MessageSpool.
GetEnvironmentVariable	Permite obtener el valor de cada variable de entorno del sistema, como la ruta de instalación de Windows o el directorio Windows\System32.
Info	Permite obtener la información de ensamblado de la aplicación, como versión, descripción del programa y copyright.
MinimumSplashScreenDisplay	Sirve para especificar el tiempo mínimo durante el cual mostraremos una pantalla de inicio a nuestra aplicación.
SaveMySettingsOnExit	Permite cambiar la manera de guardar la configuración del entorno antes de salir de la aplicación. Se combina con el método Save de My.Settings.

MY.APPLICATION	DESCRIPCIÓN
Shutdown	Permite controlar el cierre de la aplicación provocado por un evento Shutdown que se genere desde otro programa.
SplashScreen	Permite establecer u obtener la pantalla de inicio de nuestra aplicación.
Startup	Evento que se produce al iniciar la aplicación. Con él se puede tener acceso a parámetros adicionales especificados en una línea de comandos.
UICulture	Devuelve un objeto CultureInfo que muestra la referencia cultural utilizada en un subproceso.
UNhandledException	Muestra la información de la aplicación cuando ocurre una excepción no controlada.

Tabla 3. *My.Application posee los objetos necesarios para desarrollar aplicaciones multilenguaje sin gran esfuerzo.*

Veamos a continuación cómo **My.application** es de suma utilidad para el desarrollo de aplicaciones. Iniciemos un nuevo desarrollo **Windows Forms** al cual llamaremos **VariablesDeEntorno**. Le agregamos siete **Labels** y siete **TextBox**, alineados uno debajo del otro. Luego, en el evento **Form_Load** escribimos el siguiente código:

```
TextBox1.Text = My.Application.GetEnvironmentVariable("SYSTEMDRIVE").ToString
TextBox2.Text = My.Application.GetEnvironmentVariable("WINDIR").ToString
TextBox3.Text = My.Application.GetEnvironmentVariable("PATHEXT").ToString
TextBox4.Text = My.Application.GetEnvironmentVariable("TEMP").ToString 'Es
válido "TMP" también
TextBox5.Text = My.Application.GetEnvironmentVariable("OS").ToString
TextBox6.Text = My.Application.GetEnvironmentVariable("COMPUTERNAME").ToString
TextBox7.Text = My.Application.GetEnvironmentVariable("APPDATA").ToString
```

Ejecutamos el proyecto y veremos toda la información obtenida en él.

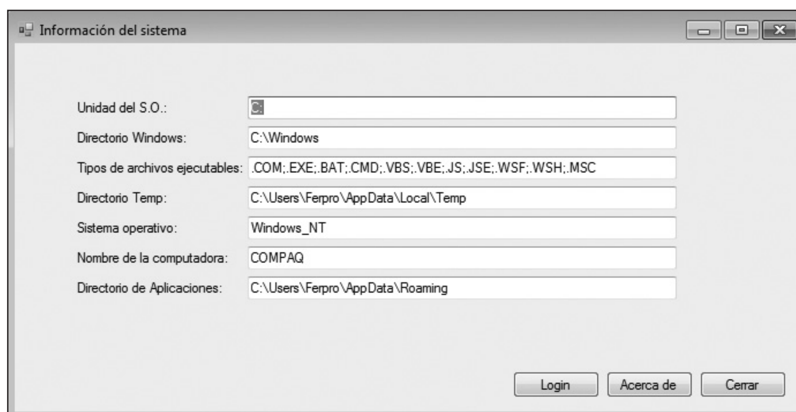


Figura 4. *Gracias a las **variables de entorno** propias del sistema operativo, podemos obtener fácilmente la información de arquitectura y del entorno de usuario de **Windows**.*

Vamos a añadir a continuación dos botones en el extremo inferior izquierdo de la ventana: uno cerrará el programa y el otro mostrará información adicional. Para no crear la ventana **Acerca de** desde cero, aprovechemos los **Templates** que nos ofrece Visual Basic. Agregamos un nuevo **Windows Form** del tipo **Cuadro Acerca De**. En uno de los botones, que etiquetaremos como "**Acerca de**", incorporamos el siguiente código al evento **Click**:

```
AboutBox1.ShowDialog()
```

Ahora agregamos la información adicional a nuestra aplicación antes de distribuirla. Para hacerlo, vamos al menú **Proyecto/Propiedades de...**. Se abrirá una nueva ventana, en la cual podremos especificar el nombre de la aplicación, su versión, una descripción detallada de su funcionamiento, el copyright, su icono, con qué formulario iniciará, y otra información adicional referida al ejecutable que distribuiremos cuando finalicemos cada proyecto.

PESTAÑA	DESCRIPCIÓN	VALOR
Aplicación	Nombre del ensamblado	Variables De Entorno
	Tipo de aplicación	Aplicación de Windows Forms
	Icono	Especifiquemos un icono a gusto...
	Formulario de inicio	Form1
Botón Información del ensamblado	Título	Aplicación de variables de entorno
	Descripción	Permite ver las variables de entorno utilizadas en Windows
	Compañía	Aquí va nuestro nombre
	Producto	VariablesDeEntorno
	Copyright	Año y nuestro nombre
	Marca comercial	Nuestro nombre o el de nuestra empresa
Publicar	Versión de la publicación	1.0.0.0
	Requisitos previos	Tildar Microsoft .NET framework 4.0 Client Profile

***Tabla 4.** Las propiedades de nuestro proyecto permitirán agregar toda la información necesaria, antes de la compilación.*

PUBLICAR NUESTRA APLICACIÓN

En el menú **Proyecto/Publicar**, encontramos un asistente que nos guía en el proceso para generar el instalador de la aplicación. Es breve y podemos configurarlo de tal manera que se distribuya como un instalador de disco o como una app instalable a través de la Web. También podemos agregar el framework .NET junto con nuestra aplicación.

Luego de hacer esto, cerramos la pestaña de propiedades y, desde el menú **Generar**, optamos por **VariablesDeEntorno**. Se creará el ensamblado de nuestra aplicación, el cual podemos ver en la ruta **CarpetaDelProyecto\NombreDelProyecto\Bin\Release\nombredelejecutable.exe**. Ejecutamos nuestro proyecto y utilizamos el botón **Acerca de** para ver la información de la aplicación, copyright y versión.

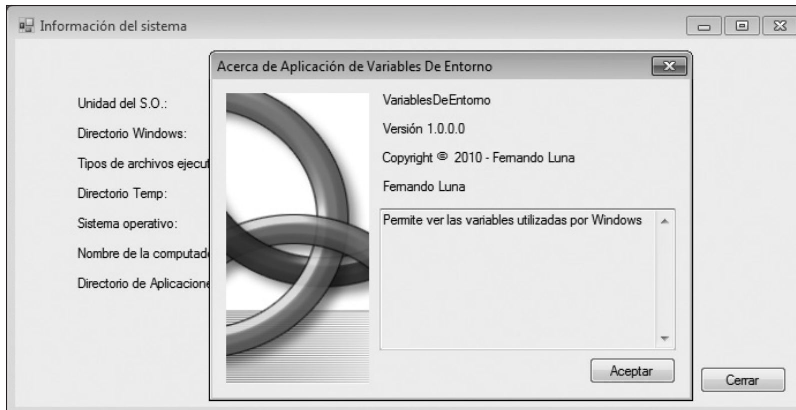


***Figura 5.** Nuestra primera aplicación ensamblada, desde la cual vemos las variables de entorno de Windows, como también el cuadro de diálogo con la información del programa.*

My.User

My.User permite obtener información sobre el usuario actual de la sesión de Windows. En sistemas de misión crítica, suelen utilizarse archivos **.LOG** para saber el nombre de usuario y la máquina desde donde se llevaron a cabo ciertas operaciones. Aunque el usuario de un sistema propietario difiere del nombre de usuario utilizado en Windows para iniciar la sesión en red, este último puede obtenerse desde **My.User**.
A nuestro proyecto anterior, vamos a agregarle un botón con la etiqueta "**Login**", y a continuación, un **Windows Forms** del tipo **Inicio de sesión**. A nuestro formulario le añadimos un **Label** entre el **textBox** nombre de usuario y el **textBox** contraseña.
En el evento **Form_Load** del formulario **LoginForm1** agregamos el siguiente código:

```
Me.UsernameTextBox.Text = My.User.Name
Label1.Text =
My.User.IsInRole(ApplicationServices.BuiltInRole.Administrator).ToString
```

My.User.Name nos devolverá el nombre de usuario que inició sesión en Windows. Si la computadora pertenece a una red con **Dominio**, el nombre de usuario estará precedido por el nombre de dominio. **Label1** mostrará, a través de **My.User.IsInRole()**, si el usuario tiene privilegios de **Administrador** o no.

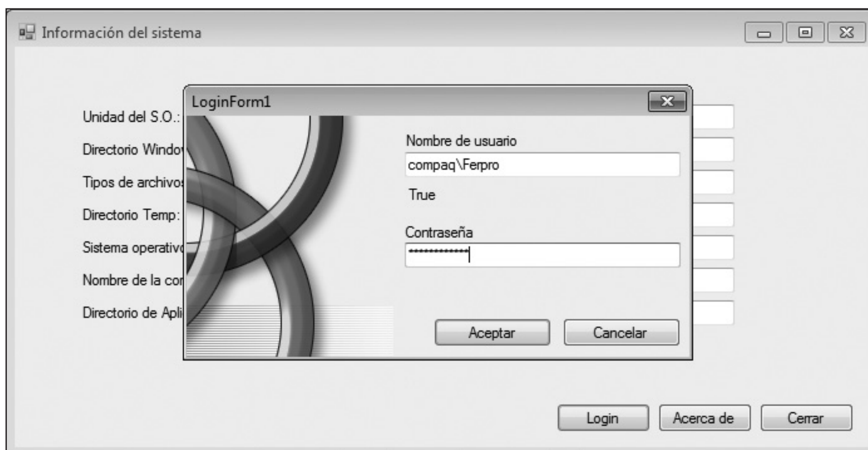


***Figura 6.** Con My.User.Name podemos predeterminar que el inicio de sesión de nuestras aplicaciones sea con el mismo usuario que inició sesión en la computadora.*

My.Settings

El namespace **My.Settings** permite acceder a la configuración de nuestra aplicación y guardarla de manera dinámica. A través de él, podemos establecer una estructura en la interfaz o de valores por defecto dentro de ella. El diseñador de configuración es clave para agregar o quitar valores del programa. La estructura de valores que debemos respetar es la siguiente:

- **Nombre**: especifica el nombre de configuración de nuestra aplicación.
- **Tipo**: especifica el tipo de dato por almacenar (**string**, **single**, **integer**, **boolean**).
- **Ámbito**: indica si es solo lectura o lectura-escritura.
- **Valor**: indica el valor por guardar o el valor por defecto a utilizar para la configuración del namespace **My.settings**.

Iniciamos una nueva **Aplicación de Windows Form** a la cual guardamos con el nombre de **ConexionRemota**. Agregamos tres **textBox**, tres **Label** y un **button**. Un **Label** será "**Dirección IP:**", el otro será "**Usuario:**" y el último será "**Clave:**". El botón será "**Guardar Configuración**".



INDENTACIÓN

La indentación (o tabulación) en el código fuente siempre fue algo primordial para poder mejorar la lectura de cada instrucción o algoritmo. Visual Basic, desde su versión 4.0, trae indentación automática a cuatro espacios. Si bien es muy cómodo utilizarla, podemos personalizarla de acuerdo con nuestra necesidad o costumbre.

Guardamos todo el proyecto y, luego, desde el menú **Proyecto**, elegimos **Propiedades de ConexionRemota**. Se abrirá la ventana anterior, desde donde configuramos las opciones básicas de la aplicación para compilarla. Vamos a la pestaña **Configuración**, donde creamos tres variables con sus respectivos valores, tal como vemos en la **Figura 7**.

	Nombre	Tipo	Ámbito	Valor
▸	direccionip	String	Usuario	200.240.1.12
	usuario	String	Usuario	root
	clave	String	Usuario	miclave
*				

***Figura 7.** La opción Ámbito de My.Settings nos permite establecer como usuario una configuración de lectura-escritura, y como aplicación, una configuración de solo lectura.*

Una vez creadas estas variables, cerramos la pestaña de propiedades y escribimos el siguiente código en **Form1**:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        My.Settings.direccionip = TextBox1.Text
        My.Settings.usuario = TextBox2.Text
        My.Settings.clave = TextBox3.Text
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        TextBox1.Text = My.Settings.direccionip.ToString
        TextBox2.Text = My.Settings.usuario.ToString
        TextBox3.Text = My.Settings.clave.ToString
    End Sub
```



OPTIMIZACIÓN CON MY.SETTINGS

My.Settings nos permite almacenar la configuración de cada control, formulario y objeto de nuestra aplicación en un archivo propio, para así no tener que utilizar el Registro de Windows. Con esto ganamos tiempo de carga como en la aplicación de los valores, y a su vez, tenemos la ventaja de disponer de una app portable siempre configurada.

Guardamos otra vez nuestro proyecto y lo ejecutamos con **F5**. Por defecto, **Form1** trae los valores establecidos en la configuración predefinida. Si cambiamos algún valor y presionamos el botón **Guardar Configuración**, quedarán almacenados como la nueva configuración de la aplicación. Para probar esta opción, detengamos nuestro proyecto y volvamos a iniciarlo.



Figura 8. *Así de simple podemos almacenar la configuración por defecto de nuestro programa y volver a usarla.*

CONTROLES

A lo largo del libro, hemos utilizado controles para crear nuestras aplicaciones de ejemplo, pero aún no profundizamos en las funciones y usos de ellos en nuestro entorno de desarrollo. A continuación, haremos una introducción a los controles y sus funciones dentro de aplicaciones Visual Basic .NET.
Los **controles** son las herramientas gráficas que utilizamos para construir la interfaz de usuario de la mayoría de las aplicaciones realizadas en el entorno de programación. Los mismos se localizan en el **Cuadro de herramientas** de Visual Basic, y del resto de los lenguajes que componen la suite Visual Studio. Se agrupan en **Categorías**, las cuales nos permiten encontrarlos rápidamente para agregarlos a diferentes proyectos.

Controles comunes

Los **controles comunes** son los que habitualmente encontramos en cualquier aplicación Windows. No todos son iguales ni cumplen una función **visible** dentro de los programas. Algunos de ellos pueden ser **invisibles**, o su forma inicial en **modo Desarrollo** puede ser distinta de la que toman en el **modo Ejecución** de las aplicaciones. Los controles pueden agregarse a través del Cuadro de herramientas y, también, creados por código.

Eventos de cada control

La mayoría de los controles poseen eventos. Los eventos contienen porciones de código o algoritmos que se ejecutan ante una determinada acción del usuario que utiliza la aplicación o de una acción que realice el sistema operativo.

Vamos a crear un nuevo proyecto del tipo **Windows Forms** llamado **Controles**. A continuación, desde la Barra de herramientas agregamos un control **Button**, y dentro del evento **Click**, escribimos el siguiente código:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim textBox1 As New TextBox()
        Dim textBox2 As New TextBox()
        textBox1.Location = New Point(50, 50)
        textBox2.Location = New Point(50, 100)
        Me.Controls.Add(textBox1)
        Me.Controls.Add(textBox2)
    End Sub
```

Para probar esto, ejecutamos el proyecto y hacemos clic en **Button1**. El resultado será el que muestra la **Figura 9**.

```
Public Class Form2

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Loa
        Dim textBox1 As New Windows.Forms.TextBox
        Dim textBox2 As New Windows.Forms.TextBox

        Me.Controls.Add(textBox1)
        Me.Controls.Add(textBox2)
        textBox1.Location = New Point(50, 50)
        textBox2.Location = New Point(50, 80)

    End Sub
End Class
```

Figura 9. *En este ejemplo vemos la creación de controles en un formulario Windows a través de código. Con pocas líneas, podemos generar controles dinámicos en una aplicación.*



PROPIEDADES Y EVENTOS POR DEFECTO

La mayoría de los controles tienen relación entre su evento predefinido y su propiedad más utilizada. Para el caso de **TextBox**, su propiedad más utilizada es **Text** y su evento por defecto es **TextChanged**, mientras que para **Button**, su evento es **click()**.

Cada control proporciona eventos, métodos y propiedades diversas, dependiendo de su funcionalidad. Veamos a continuación los controles y funciones del entorno.

Button

Representa los botones en las aplicaciones. Sus eventos más utilizados son los que se listan en la **Tabla 5**.

EVENTOS	DESCRIPCIÓN
Click, Enter, MouseUp, MouseDown, MouseMove, MouseHover	Son los eventos más relevantes de este control. En el nombre de cada uno se describe la acción que se está realizando.
KeyPress, KeyUp, KeyDown	Estos eventos permiten detectar la información proporcionada por el teclado.

***Tabla 5.** Eventos y propiedades del control Button.*

Label, LinkLabel

Estos controles permiten mostrar un texto. **LinkLabel** se diferencia de **Label** porque el texto mostrado se hace como si fuera un **hipervínculo**, con lo cual nos permite indicar al igual que en un website, que haciendo **clic** en él y con código mediante, iremos hacia otro **formulario** o **página web**. Sus propiedades y eventos más comunes se muestran en la **Tabla 6**.

PROPIEDAD	DESCRIPCIÓN
Text	Permite especificar el texto que mostrará.
TextAlign	Permite alinear el texto contenido.
Autosize	Permite tomar un tamaño que se ajuste o no al texto contenido.
Image	Permite mostrar una imagen.
Font	Permite especificar una fuente, tamaño y estilo con el que se mostrará el texto.
Evento	Descripción
No posee ningún evento especial. Hereda los eventos de cualquier otra clase o control, tal como vimos en Button.	

***Tabla 6.** Eventos y propiedades de los controles Label y LinkLabel.*



PROPIEDAD PASSWORDCHAR DE TEXTBOX

La propiedad **PasswordChar** solo permite elegir un carácter que se mostrará cuando se ingresa un dato en la caja de texto. Esto únicamente se aplica al modo visual, no realiza ningún tipo de encriptación. Para encriptar cualquier dato en nuestras aplicaciones debemos aplicar las clases del namespace **System.Security.Cryptograpy**.



Figura 10. *Controles Label y LinkLabel representados en un **Form**.*

TextBox

Este control permite el ingreso de datos mediante el teclado. Posee propiedades y eventos destinados a tener control sobre su contenido, como podemos ver en la **Tabla 7**.

PROPIEDAD	DESCRIPCIÓN
Text	Texto que mostrará.
TextAlign	Permite alinear el texto contenido.
MaxLength	Se puede indicar la cantidad máxima de caracteres. El valor 0 permite ingresar texto ilimitado.
CharacterCasing	Procesa el texto ingresado, para transformarlo en mayúscula, minúscula, o dejarlo tal como fue escrito.
PasswordChar	Permite especificar un carácter que se visualizará en vez del texto ingresado, como en los casos de ingreso de contraseña.
Multiline	Estableciendo esta propiedad en True, podremos escribir texto en varias líneas.
ScrollBars	Se aplica si la propiedad Multiline es True para mostrar barras de scroll. Puede ser Scroll vertical, horizontal o ambos.
Readonly	En True, no permitirá modificar el texto que muestra.
TextChanged	Evento que se produce cuando el texto contenido cambia.
KeyPress	Evento que recibe como parámetro adicional el código de tecla presionada, para realizar una acción.
KeyUp / KeyDown	Similar al evento KeyPress, pero el primero detecta cuando la tecla se soltó, mientras que el segundo detecta cuando se la presionó.

Tabla 7. *Eventos y propiedades del control textBox.*

Iniciamos un nuevo proyecto **Windows Forms** llamado **ControlDeTexto**. Agregamos en el form correspondiente un **Label** y un **textBox**, y luego, añadimos el siguiente código en cada uno de los eventos listados:

```
Public Class Form1
    Dim k As Long
    Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
        If e.KeyChar = Microsoft.VisualBasic.ChrW(Keys.Return) Then
            k = k + 1
            Label2.Text = “Cantidad de Enter presionados: “ & k
        End If
    End Sub

    Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles TextBox1.TextChanged
        Label1.Text = “Caracteres: “ & Len(TextBox1.Text)
    End Sub
End Class
```

La función **Len()** analiza cuántos caracteres hay en el contenido de **TextBox1** y lo muestra en **Label1**. **Label2** refleja el resultado analizado en el evento **textBox1_KeyPress()**, donde obtenemos cuántas veces hemos presionado la tecla **Enter** (**Return**), y sumando a la variable **k 1**por cada **Enter** presionado, informamos la cantidad en pantalla el total de esta acción. Ejecutemos el proyecto y escribamos texto en él, presionando **Enter** reiteradas veces.

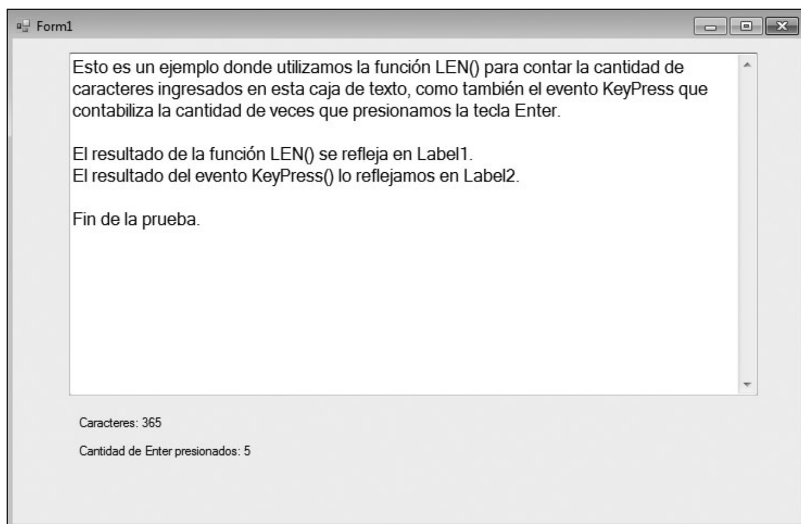


Figura 11. *Los controles Label y textBox en acción a través de este ejemplo.*

ListBox y ComboBox

Ambos controles funcionan de manera similar en cuanto a sus propiedades y eventos. **ComboBox** es una caja de texto desplegable que permite mostrar y seleccionar un ítem entre todos los ingresados. **ListBox** es similar a él, aunque muestra los ítem de manera consecutiva. Las propiedades más comunes pueden verse en la **Tabla 8**.

PROPIEDAD COMBOBOX	DESCRIPCIÓN
DropDownStyle	Indica qué estilo de selección mostrará. El más común es DropDownList.
Items	Agrupa dentro de una colección la cantidad de elementos que contiene.
SelectedIndex	Permite obtener o indicar qué elemento de la colección de ítem está seleccionado, y devuelve su número de índice.
Sorted	Permite establecer un ordenamiento de los elementos.
SelectedItem	El elemento seleccionado por defecto.
Items	Colección que contiene los elementos que el usuario puede ver y seleccionar.
SelectionMode	Permite seleccionar uno o más elementos.
MultiColumn	Permite establecer un listBox con más de una columna de datos.
SelectedItems	Permite obtener qué elementos fueron seleccionados por el usuario, solo si SelectionMode se estableció como Múltiple.

***Tabla 8.** Propiedades más utilizadas de ComboBox y ListBox.*

En ambos controles utilizamos la función **control.ítems.add()** para agregar contenido. A través del evento **Control_SelectedIndexChanged()** detectamos si se ha elegido una opción distinta. Vamos a crear un nuevo proyecto **Windows Forms**, al que agregamos un **listBox** y un **ComboBox**. A continuación, escribimos el siguiente código:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ComboBox1.Items.Add("Libros de programación")
        ComboBox1.Items.Add("Libros de literatura")
        ComboBox1.Items.Add("Cómic Books")
```



DIFERENCIAS ENTRE TEXTBOX Y MASKEDTEXTBOX

MaskedTextBox permite utilizar dicho control como si fuese un TextBox común, con la ventaja de poder formatear su contenido de manera predeterminada a través de la propiedad **Mask**. Podemos utilizar algún formato predeterminado o crear el nuestro.

```
    End Sub

Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
        Select Case ComboBox1.Text
            Case "Libros de programación"
                ListBox1.Items.Clear()
                ListBox1.Items.Add("Visual Basic 2010")
                ListBox1.Items.Add("Visual C# 2010")
                ListBox1.Items.Add("Visual C++")
            Case "Libros de literatura"
                ListBox1.Items.Clear()
                ListBox1.Items.Add("La caída de los gigantes")
                ListBox1.Items.Add("Las legiones malditas")
                ListBox1.Items.Add("El sueño del celta")
            Case "Cómic Books"
                ListBox1.Items.Clear()
                ListBox1.Items.Add("Spider-man")
                ListBox1.Items.Add("Superman")
                ListBox1.Items.Add("Batman")
        End Select
    End Sub
```

El resultado de nuestra aplicación será el que se muestra en el **Figura 12**.

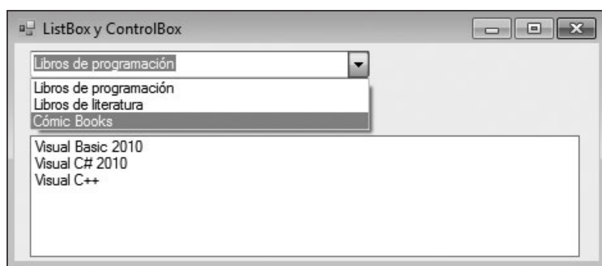


***Figura 12.** ListBox y ComboBox utilizando ítems.Add() y el evento SelectedIndexChanged() para detectar el nuevo elemento seleccionado.*

DateTimePicker y MonthCalendar

Estos controles permiten manipular datos a través de las fechas. **DateTimePicker** funciona como un **ComboBox** mostrando un calendario, mientras que **MonthCalendar** muestra un calendario completo en pantalla. **DateTimePicker**, como su nombre lo

indica, entrega como valor la fecha y hora, mientras que **MonthCalendar** brinda solamente la fecha seleccionada.
Estos controles tienen propiedades variadas que permiten establecer parámetros sobre sí mismos para lograr un mayor control de sus propiedades y eventos.

PROPIEDADES DATETIMEPICKER	
PROPIEDAD	DESCRIPCIÓN
Value	Permite obtener o indicar un valor fecha del calendario.
MinDate	Especifica cuál es la fecha mínima que se puede mostrar.
MaxDate	Especifica cuál es la fecha máxima que se puede mostrar.
Format	Establece cuál es el formato por mostrar en la caja de texto del control.

Tabla 9. *Las propiedades más utilizadas de **DateTimePicker**.*

A través del evento **DateTimePicker_ValueChanged()** podemos detectar la fecha seleccionada en dicho calendario.

PROPIEDADES MONTHCALENDAR	
PROPIEDAD	DESCRIPCIÓN
SelectionRange	Devuelve los valores mínimos y máximos de un rango de fechas seleccionado en el calendario.
MinDate	Misma función que DateTimePicker.
MaxDate	Misma función que DateTimePicker.
CalendarDimensions	Establece los meses que mostrará el calendario.

Tabla 10. *Las propiedades más utilizadas de **MonthCalendar**.*

Detectamos la fecha seleccionada a través del evento **DateSelected()**, y el resultado se obtiene usando el valor **e.Start.Date**.

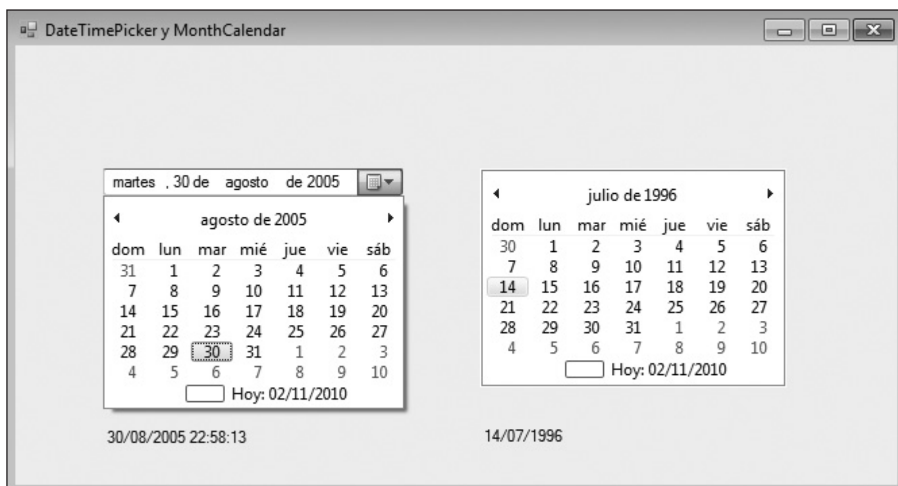


Figura 13. *Los controles calendario en acción.*

RadioButton y CheckBox

Estos controles básicos permiten que los usuarios establezcan una selección de opciones. **CheckBox** puede interactuar con otros controles similares a sí mismo, y admite marcar varios al mismo tiempo. **RadioButton** permite marcar uno solo entre varios de su tipo, siempre que todos se encuentren dentro de un mismo contenedor. La propiedad **Text** indica el texto que se mostrará junto a la opción de lista. Su propiedad **Checked** indica o establece si el control fue marcado o seleccionado.

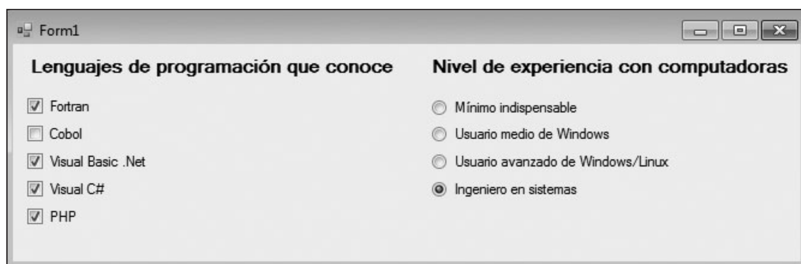


Figura 14. *En ejemplo de un formulario/encuesta con controles CheckBox y RadioButton.*

Otros controles comunes

- **NotifyIcon**: es un control no visible en nuestra aplicación. Permite generar un icono en la barra de tareas de Windows con funciones u opciones que nosotros establezcamos. También soporta menú contextual.
- **PictureBox**: permite mostrar una imagen seleccionada desde nuestro disco rígido, como de Internet o algún recurso incluido en la aplicación que contenga una colección de imágenes. Soporta los formatos más comunes de imágenes (**GIF**, **PNG**, **JPG**, **TIF**, **BMP**, **ICO**) y también permite usar imágenes con transparencias.
- **ProgressBar**: muestra una **barra de progreso**, generalmente combinada con la ejecución de algún proceso complejo en nuestro sistema. Sus propiedades **Minimun** y **Maximun** establecen los parámetros mínimos y máximos de la barra, mientras que la propiedad **Value** setea el porcentaje de progreso.
- **ToolTip**: este **control no visible** permite establecer una etiqueta con un **texto descriptivo** sobre los controles de la aplicación, cuando posicionamos el puntero del mouse sobre alguno de ellos.
- **ListView**: permite mostrar en forma de lista una jerarquía de elementos. Estos pueden ser una colección de objetos, **registros de una base de datos**, **archivos/carpetas** del disco o **unidades** de la computadora.
- **TreeView**: muestra una colección de objetos en forma de **árbol**, partiendo de una raíz, y conteniendo **nodos** y **subnodos** según nuestra necesidad. El ejemplo más común donde encontramos el uso de **TreeView** y **ListView** es en **Windows Explorer** y en el panel de control, entre otros.

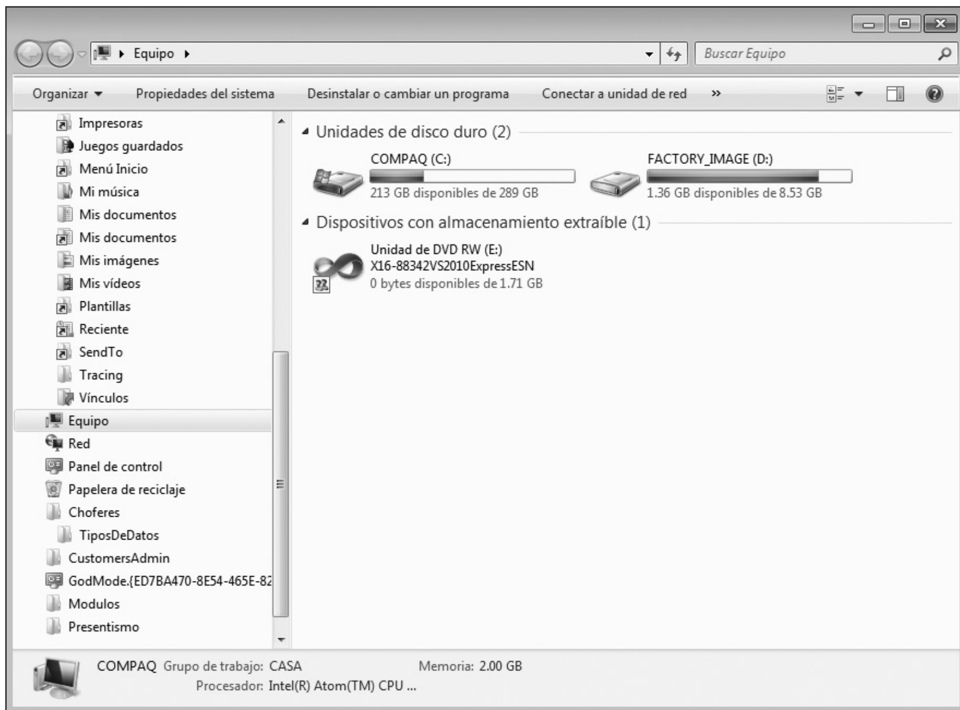


***Figura 15.** **Windows Explorer** agrupa los controles **TreeView** y **ListView** para mostrar las unidades, carpetas y archivos de nuestra computadora.*

Controles contenedores

Los **controles contenedores** permiten que nuestra aplicación incluya varios controles de cualquier tipo dentro de sí misma, y así organizarlos mejor en la pantalla. Otras ventajas de estos controles es poder establecer una propiedad **visible** de manera grupal, como una propiedad **Enabled** también de manera grupal, sin necesidad de tener que cambiar dichas propiedades por cada uno de los controles.

- **GroupBox**: permite sectorizar información común dentro de una ventana. Es útil para agrupar controles **RatioButton** y **CheckBox**.



CONTROLES Y EVENTOS

A partir de **RadioButton** y **CheckBox**, veremos un pantallazo generalizado de cada control. Hay muchos otros que pasamos por alto, pero mencionar cada uno de ellos requeriría dedicar un libro completo. Invitamos a los lectores a que utilicen la ayuda proporcionada por la herramienta para expandir aún más sus conocimientos.

- **Panel**: posee bordes transparentes, con lo cual podemos distribuir controles en un Form a través de sus diferentes regiones. Cuenta con más opciones que **GroupBox**, como **Dock** y **Anchor**, que nos ayudan a posicionar fácilmente el resto de los controles contenidos, y admite **scrollbars** para mostrar parte de su contenido en pantalla.
- **TabControl**: agrupa varios controles, que pueden organizarse a través de solapas (**TabPages**). Cada una puede configurarse de manera independiente. Es muy útil para organizar mucha información por categorías y cuando un **Form** de pequeñas dimensiones debe contener muchos controles.

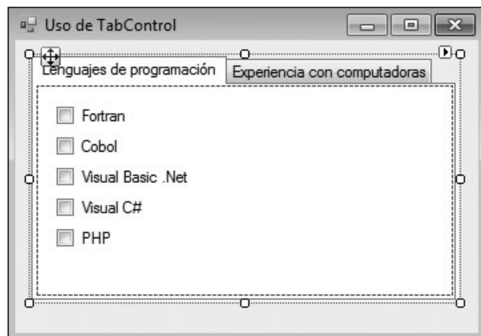


Figura 16. *TabControl nos permite agrupar controles en pestañas de manera fácil y práctica.*

Controles de menús y barras de herramientas

Desde la aparición de .NET, las herramientas de programación de **Visual Studio** poseen un mejor soporte para la creación de **menús** y **barras de herramientas**, como también de **barras de estado**. Así, se ha logrado que los elementos de cada una de ellas sean verdaderos objetos independientes entre sí. Estos elementos son muy comunes y los encontramos en cualquier aplicación que corra sobre Windows.

- **ToolStripContainer**: es un objeto contenedor que divide nuestro **Form** en **tres paneles horizontales**. El superior agrupa la **barra de herramientas** de la aplicación, como también la **barra de menús** si es que la hay. El panel del medio permite



SPLITCONTAINER Y FLOWLAYOUTPANEL

Permiten organizar otros componentes en un Form. **SplitContainer** los organiza a manera de grilla de datos, mientras que **FlowLayoutPanel** lo hace como si fuera un diagrama de flujo. Con todas estas ventajas, ahorramos decenas de líneas de código que nos permitirán reestructurar los controles en pantalla ante un cambio de dimensión del Form.

establecer todo el **contenido de la ventana** con el que el usuario suele interactuar. El inferior se utiliza para agrupar una **barra de tareas**.

- **MenuStrip**: con él podemos establecer los menús que tendrá nuestra aplicación.
- **ContextMenuStrip**: con la misma funcionalidad que **MenuStrip**, permite establecer menús en nuestros desarrollos, que solo serán visualizados a través del botón derecho del mouse, previo código que lo invoque.
- **ToolStrip**: permite crear una barra de herramientas en la aplicación, que podrá contener varios botones, personalizables al máximo en cuanto a estética y funciones, al igual que las barras de cualquier aplicación.

Vamos a iniciar un nuevo proyecto **Windows Forms** y a guardarlo con el nombre **ToolbarsMenus**. A continuación, agregaremos un nuevo elemento del tipo **Windows Forms**. De la lista de **templates** seleccionamos **Formulario primario MDI**. El resultado será el que se muestra en la **Figura 17**.

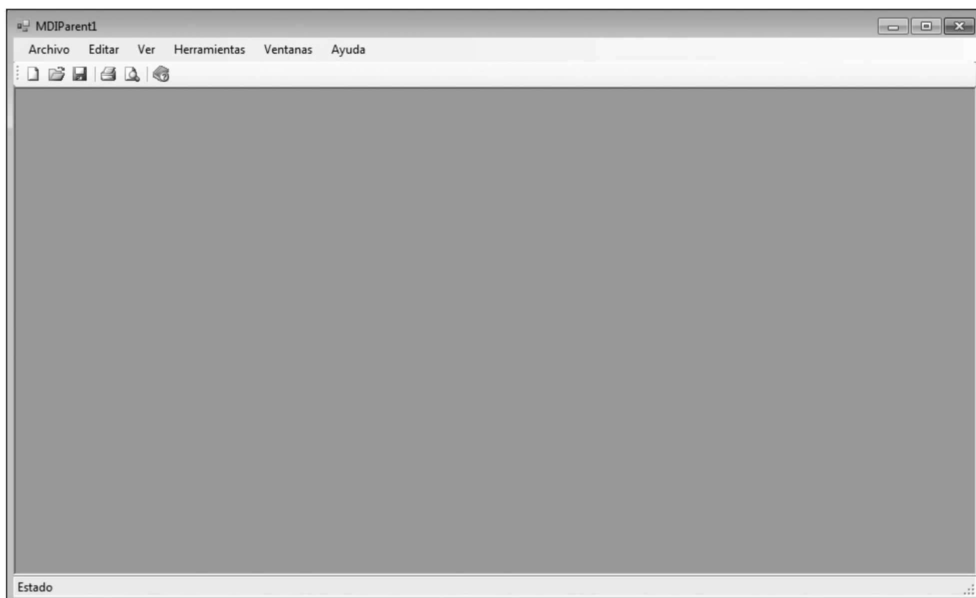


***Figura 17.** MDIParent1 es un FormMDI con las opciones comunes de menús y barras de herramientas ya creadas.*

Sin haber escrito una sola línea de código, pudimos crear un formulario completo con menús y barras de herramientas. Con un poco de paciencia, podremos personalizar dicha ventana para que muestre los menús y botones según nuestra necesidad.
La creación de menús ya no se limita a simples listas desplegables con posibilidad de incluir submenús. Como los controles utilizados son objetos en un 100%, dentro de cualquier elemento de un menú se puede establecer un subelemento del tipo **MenuItem**, **TextBox** o **ComboBox**.

Una vez que agregamos el **Form MDIParent1** a nuestro proyecto, ya podemos eliminar **Form1**, presionando para ello el botón derecho del mouse sobre él y seleccionando **Menú contextual/Eliminar**.
En la barra de herramientas encontramos la opción de agregar más elementos **Button** a nuestro control. Dichos ítem pueden variar entre **Buttons**, **Label**, **SplitButton**, **DropDownButtons**, **ComboBox**, **TextBox** y **ProgressBar**. Al igual que otros controles, poseen los mismos eventos para trabajar con ellos.

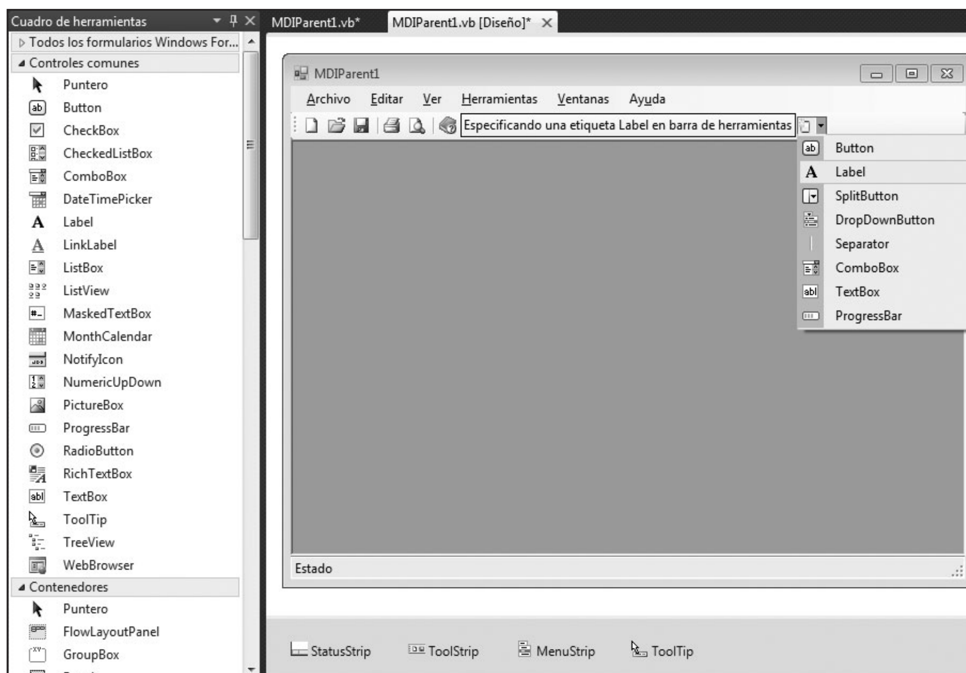


Figura 18. *Las opciones de personalización de una toolbar o un menú ahora son infinitas.*

Controles de acceso a datos

Los **controles de acceso** a datos nos permiten establecer conexión con una base de datos para recuperar los registros de las **tablas** y trabajar con ellos, así como



EVENTOS DE MENÚS

Los menús poseen la mayoría de los eventos encontrados en cualquier otro control de Visual Basic. Dentro del evento **Click()** podemos especificar un algoritmo por procesar, y dentro del menú **MouseHover()**, indicar que en la barra de tareas de la aplicación se muestre un mensaje con la acción del menú donde nos encontramos, tal como lo hace cualquier aplicación de Windows.

también para mostrar **vistas de datos** y realizar operaciones como **Alta**, **Baja** y **Modificación** de registros, o llamadas a **Stored Procedures** que permitan efectuar otras operaciones. En el capítulo que abordamos el tema bases de datos, veremos en profundidad cada uno de ellos.

Convenciones para nombrar los controles

Dentro de todo grupo de trabajo que comparte determinados desarrollos de software, suelen establecerse convenciones de nombres para cada objeto y variables que componen el sistema. Esto permite que cualquier programador, con solo leer un nombre de objeto dentro del código, detecte si pertenece a una clase objeto o se está haciendo referencia a un control.
Hasta ahora en los ejercicios hemos dejado el nombre de cada control tal como lo establece Visual Basic al crear un nuevo objeto en pantalla. Esto nos permite realizar más rápidamente los ejemplos del libro, y también familiarizarnos con el nombre común que tienen los controles. A partir de ahora estableceremos en lo posible, una convención de nombres de control.
En la **Tabla 11** vemos una lista que puede orientarnos para poner nombres comunes a los controles en nuestros desarrollos.

CONTROL	NOMBRE TENTATIVO	EJEMPLO
Button	btnNombreDeAccion	btnNuevoDocumento
Label	lblNombre	lblHipervinculo
TextBox	txtNombre	txtApellido
CheckBox	chkControl	chkLenguajeVBNET
RadioButton	rdbControl	rdbExperienciaBasica
ListView	lvControl	lvArchivos
MonthCalendar	mCalNombre	mCalAgenda
DataGrid	dgControl	dgClientes
PictureBox	picImagen	picFotoPerfilDerecho

Tabla 11. *Una de las posibles convenciones de nombre para los controles en nuestros proyectos.*



INTERACCIÓN CON EL USUARIO

Visual Basic .NET pone cuadros de diálogo que permiten interactuar a las aplicaciones con el usuario a través de mensajes o avisos. Estos cuadros pueden realizar una acción u otra dependiendo de la elección que el usuario haya hecho.

MessageBox

La clase **MessageBox** permite generar un cuadro de diálogo a través del cual el usuario recibe un mensaje. Este puede ser a modo informativo o a modo de cuestión. Puede contener un icono que ilustre el tipo de mensaje (**aviso**, **advertencia**, **error**, **pregunta**, **exclamación**), y también tener botones diferentes a través de los cuales se lleven a cabo distintas acciones (**Aceptar**, **Cancelar**, **Sí**, **No**, **Reintentar**).

Iniciemos un nuevo proyecto **Windows Forms**, llamado **CuadroDeMensaje**, al que le agregaremos un control **Button** y, dentro del evento **Click**, el siguiente código:

```
Dim msg As String, res As DialogResult
msg = "Usted ha presionado un botón. Dependiendo de la opción que elija en
este cuadro "
msg = msg & "de diáologo, se abrirán distintas aplicaciones."
msg = msg & "Presione SI para iniciar Notepad, Presione NO para iniciar
Windows Explorer."
res = MessageBox.Show(msg, "Cuadro de Mensaje", MessageBoxButtons.YesNo
Cancel, MessageBoxIcon.Information)
         If res = System.Windows.Forms.DialogResult.Yes Then

'Presionó SI, entonces abro el Block de notas

              Shell("notepad.exe")
         ElseIf res = System.Windows.Forms.DialogResult.No Then

'Presionó NO, entonces abro Windows Explorer

              Shell("explorer.exe")
         Else

'Presionó CANCEL, entonces finalizo la aplicación

              End
         End If
```

La variable **res** permite obtener el botón presionado del cuadro de diálogo. **MessageBox.Show()** muestra el cuadro de diálogo. El parámetro **MessageBoxButtons** permite establecer la combinación de botones por mostrar. **MessageBoxIcon** establece el icono que se presentará para describir el tipo de mensaje. Por último, la instrucción **Shell()** llama a la ejecución de un programa.

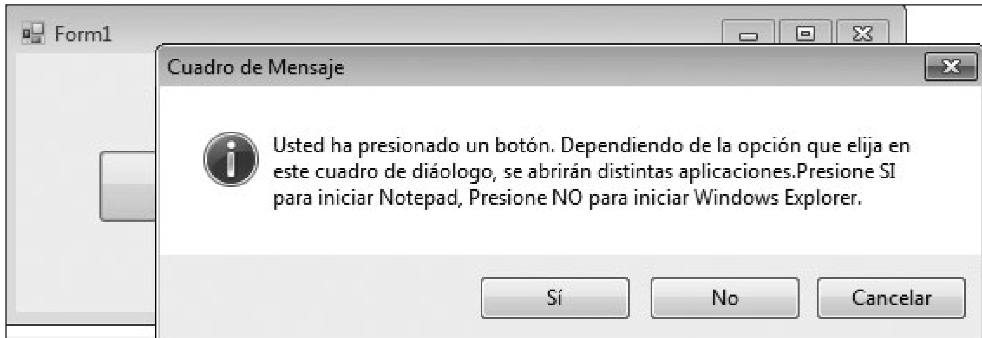


***Figura 19.** El resultado de nuestra aplicación **CuadroDeDialogo**.*

Operadores aritméticos

Los operadores aritméticos nos permiten llevar a cabo diversas operaciones matemáticas con tipos de datos numéricos en nuestras aplicaciones. En la **Tabla 12** se muestra cuáles son los principales operadores.

OPERADOR	DESCRIPCIÓN
^	Eleva un número a una potencia indicada.
*	Realiza una multiplicación entre dos números.
/	Realiza una división entre dos números.
\	Divide dos números y devuelve un entero como resultado.
Mod	Divide dos números y devuelve solo el resto.
+	Realiza una suma de dos números.
-	Realiza una resta de dos números.

***Tabla 12.** Operadores aritméticos utilizados en Visual Basic.*

Una calculadora básica

Iniciemos un nuevo proyecto **Windows Forms**, al que llamaremos **CalculadoraBasica**. A continuación, armaremos su interfaz distribuyendo un **Label** y los botones necesarios para que la calculadora quede armada. Debe quedar como se muestra en la **Figura 20**.

CONTROL	PROPIEDAD TEXT
Button1	1
Button2	2
Button3	3
Button4	4
Button5	5
Button6	6

CONTROL	PROPIEDAD TEXT
Button7	7
Button8	8
Button9	9
Button10	0
Button11	.
Button12	+
Button13	-
Button14	*
Button15	/
Button16	C
Button17	=

Tabla 13. *En esta tabla vemos todos los botones que conformarán nuestra calculadora, con su propiedad **Text**.*

```
Public Class Form1
    Dim v1 As Double, v2 As Double, res As Double, Num As String
    Dim ope As String

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        v1 = 0
        v2 = 0
        res = 0
        Num = “”
        lblScreen.Text = “0”
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Num = Num & Button1.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Num = Num & Button2.Text
        lblScreen.Text = Num
    End Sub
```

```
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        Num = Num & Button3.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        Num = Num & Button4.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        Num = Num & Button5.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
        Num = Num & Button6.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
        Num = Num & Button7.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As
```



DENOMINACIÓN DE VARIABLES

El uso de palabras clave como variables siempre requirió utilizar los primeros caracteres del tipo de dato en el nombre de las variables que declaramos. Esta consigna debe respetarse, y conviene crear nuestra propia tabla de nomenclaturas para conservar el orden y la claridad en el código.

```
System.EventArgs) Handles Button8.Click
        Num = Num & Button8.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button9.Click

        Num = Num & Button9.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button10.Click
        Num = Num & Button10.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button11_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button11.Click
        Num = Num & Button11.Text
        lblScreen.Text = Num
    End Sub

    Private Sub Button12_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button12.Click
        'Suma

        If Num <> "" Then v1 = System.Convert.ToDouble(Num)

        v2 = 0
```



ESPACIADO ENTRE CÓDIGO

La mayoría de los lenguajes de programación pasan por alto los renglones en blanco dentro del código. Este punto, al igual que la denominación de variables y la indentación, siempre quedan a gusto del programador. Una buena práctica es no dejar espacios en blanco dentro del código o completarlos con comentarios sobre el algoritmo tratado.

```
        ope = "+"
        Num = ""
    End Sub

    Private Sub Button13_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button13.Click

        'Resta

        If Num <> "" Then v1 = System.Convert.ToDouble(Num)
        v2 = 0
        ope = "-"
        Num = ""
    End Sub

    Private Sub Button15_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button15.Click

        'División

        If Num <> "" Then v1 = System.Convert.ToDouble(Num)
        v2 = 0
        ope = "/"
        Num = ""
    End Sub

    Private Sub Button14_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button14.Click

        'Multiplicación
        If Num <> "" Then v1 = System.Convert.ToDouble(Num)
```

HERENCIA DE ESTILO

Cada lenguaje de programación moderno tiene un estilo de escritura de código heredado o imitado de algún otro lenguaje que ya tiene varias décadas de uso o que ha dejado de soportarse. Visual Basic hereda su estilo de estructura de código de Fortran, mientras que C#, C++, PHP y Python heredan su estilo del clásico C y Pascal.

```
        v2 = 0
        ope = "*"
        Num = ""
    End Sub

    Private Sub Button17_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button17.Click
        If Num <> "" Then v2 = System.Convert.ToDouble(Num)
        lblScreen.Text = ejecutoCalculo(ope)
        v1 = 0
        v2 = 0
        Num = ""
        ope = ""
    End Sub
    Function ejecutoCalculo(ByVal o As String) As String
        ejecutoCalculo = 0
        Select Case o
            Case "+"
                res = v1 + v2
                ejecutoCalculo = res.ToString
            Case "-"
                res = v1 - v2
                ejecutoCalculo = res.ToString
            Case "*"
                res = v1 * v2
                ejecutoCalculo = res.ToString
            Case "/"

                res = v1 / v2
                ejecutoCalculo = res.ToString
        End Select
    End Function

    Private Sub Button16_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button16.Click
        Form1_Load(sender, e)
    End Sub
End Class
```

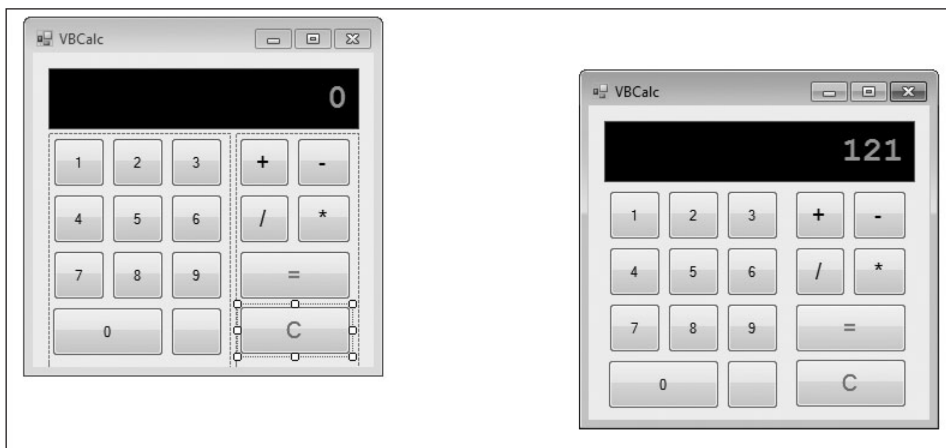


Figura 20. *Como resultado de nuestro código, ya tenemos la calculadora funcionando.*

RESUMEN

Como resultado de este capítulo, vimos algunos de los namespaces principales para trabajar en aplicaciones Windows Forms; luego realizamos un repaso por todos los controles, las propiedades y los métodos más comunes que se utilizan con ellos en Visual Basic. Por último, hicimos un recorrido por la función MessageBox y los operadores aritméticos básicos. Realizamos un proyecto de calculadora para comprender mejor la interacción de los objetos y las operaciones.



ACTIVIDADES

PREGUNTAS TEÓRICAS

1. Investigue cuáles son los namespaces reservados solamente para aplicaciones que no son Windows Forms.

2. Cree una aplicación simple de Windows Forms que permita devolver a través de un cuadro de diálogo la cultura por defecto que el sistema tiene configurada.

3. El namespace **My.User.Name** devuelve el nombre de usuario junto con el nombre de la computadora. Investigue cómo obtener sólo el nombre de usuario utilizando la función **Split()**.

4. Realice la misma acción especificada en el Punto 3, pero para obtener el Usuario y el Dominio en dos variables separadas.

5. Investigue la propiedad **FormBorderStyle** de los objetos Forms.

6. Mediante un proyecto Windows Forms intente cambiar la propiedad **FormBorderStyle** entre todas las disponibles.

7. Cree un proyecto Windows Forms e implemente la opción SI, NO, CANCELAR en la clase **MessageBox**, basándose en el ejemplo estudiado en este capítulo.

8. Investigue e implemente en un proyecto de Ejemplo la clase **InputBox**.

9. Investigue el namespace **My.Computer.Audio.Play** y cree un proyecto Windows Forms donde pueda implementar el mismo.

10. Investigue el namespace **My.Computer.Clock** y cree un proyecto donde se pueda implementarlo.

Manejo de archivos

El manejo de archivos o documentos electrónicos cumple una función importante en todo sistema adaptado a nuestra época. Visual Basic 2010 pone a nuestra disposición mecanismos que nos permiten desarrollar aplicaciones que trabajen con cualquier tipo de archivos, documentos o carpetas, manipulando la información que ellos contienen. A través de diversas funciones y espacios de nombre disponibles en .NET, podremos acceder a las operaciones más comunes con los archivos y carpetas.

ABRIR UN ARCHIVO

Muchos desarrollos requieren consultar archivos de texto para obtener o almacenar datos con algún propósito. Visual Basic, dentro del espacio de nombres **My.Computer.FileSystem**, dispone de diferentes clases que nos permitirán trabajar con archivos. Este namespace, combinado con la clase **StreamReader**, que se aloja en el namespace **System.IO**, nos dará el soporte necesario para leer los caracteres contenidos en un archivo, cualquiera sea su codificación.
Recordemos que el control **TextBox** pone a nuestra disposición las propiedades **Multiline**, que nos permiten distribuir en múltiples líneas un contenido de texto por mostrar, y la función **ScrollBars** creará **barras de scroll** automáticas para poder desplazarse en todo su contenido.
Para comprender mejor estas clases, vamos a buscar un archivo de texto en nuestra computadora que nos sirva de ejemplo y a copiarlo al **Escritorio** de **Windows**. Luego, creamos un nuevo proyecto **Aplicación de Windows Forms** llamada **TrabajoConArchivos**, a la que agregaremos los controles detallados en la **Tabla 1**.

CONTROL	NOMBRE	UBICACIÓN	FUNCIÓN
TextBox	txtArchivo	Controles comunes	Muestra el contenido del archivo abierto.
MenuStrip	MenuArchivo	Menús y barras de herramientas	Permite crear un menú con varias funciones para aplicar sobre el archivo de texto.
OpenFileDialog	Openfd	Cuadros de diálogo	Presenta un cuadro de diálogo Abrir para seleccionar el archivo de texto con el cual trabajaremos.
StatusBarStrip	stBar	Menús y barras de herramientas	Barra de estado a través de la cual presentamos varios mensajes.

Tabla1. *Estructura y controles de nuestro nuevo proyecto.*

A **txtArchivo** le modificaremos sus propiedades **Multiline = True** y **ScrollBars = Both** para poder visualizar en múltiples líneas el contenido del archivo de texto, y lograr que, a su vez, el control genere las respectivas barras de scroll para desplazarnos. A **MenuArchivo** le agregaremos los ítem que mostrará en su función de **Menú**. Para esto, hacemos clic en el icono **MenuArchivo**, ubicado al pie de **Form1**; luego, sobre la caja de texto vacía que aparece en la parte superior izquierda de **Form1** ingresamos como nombre **Archivo**. A continuación, se mostrarán submenús en los cuales podemos seguir agregando el resto del contenido. El primero será **Abrir**, el segundo será **Salir**, y entre ellos hacemos un clic con el botón derecho del mouse y, del menú contextual, elegimos la opción **Insertar/Separator**. Al final, podemos agregar teclas de acceso rápido a nuestro menú. Para lograrlo, sobre el ítem **Archivo** hacemos un clic con el botón derecho del mouse, del menú contextual seleccionamos **Editar DropDownItems**, y en el siguiente cuadro de propiedades veremos la

estructura de cada elemento de nuestra colección **MenuStrip**. Haciendo un clic sobre cualquier subítem, podremos ver sus propiedades sobre el panel derecho. Para el ítem **Abrir** ponemos la propiedad **Name = SubMenuAbrir** y **ShortCutKey = Ctrl+A**. Para el ítem **Salir** seteamos la propiedad **Name = SubMenuSalir** y **ShortCutKey = Ctrl+S** ó **CTRL+T** si nos parece más cómodo.

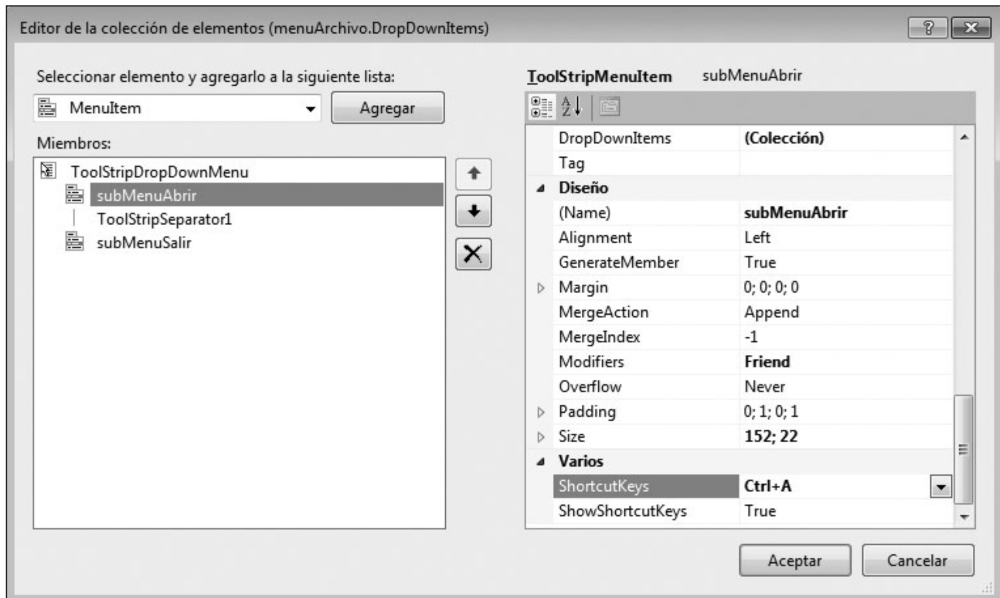


***Figura 1.** El **Editor de colección de elementos** nos permite configurar en profundidad cada elemento que compone nuestro menú **Archivo**.*

A continuación, configuraremos **OpenFileDialog1**. Seleccionamos el control con un clic y cambiamos sus propiedades por lo siguiente: **Name = Openfd**, **Filter = Archivos de texto | *.txt**, **DefaultExt = *.txt**, **Title = Abrir Archivo**. Por último, agregamos **StatusBarStrip** y seteamos su propiedad **Name = stbar**.

Una vez ajustadas las propiedades, ingresaremos el código correspondiente que nos permitirá abrir un archivo de texto. Hacemos doble clic sobre **Form1** y, antes de **Class Form1**, escribimos el siguiente código:

PERSONALIZACIÓN DE MENÚS

Los menús y submenús disponen de un nivel importante de personalización. Podemos crear atajos de teclado usando **CTRL**, **ALT** o **SHIFT** con cualquier letra disponible. También, agregar imágenes a cada ítem de los menús, cambiar las fuentes y una cantidad de funciones más. Recomendamos dedicarle un tiempo al estudio de la personalización de menús.

```
Imports System
Imports System.IO
Public Class Form1
…
```

Después de **Public Class Form1**, declaramos la variable **strTexto**:

```
…
Public Class Form1
    Dim strTexto As String
…
```

Leer datos de un archivo

Hasta ahora, simplemente hemos diseñado la interfaz de usuario, y declarado las variables y namespace que serán de utilidad para lograr nuestro propósito. Ahora veremos cómo abrir un archivo de texto y visualizarlo. A continuación, en el menú **Archivo/Abrir** hacemos doble clic e ingresamos el siguiente código:

```
strTexto = “”
Openfd.ShowDialog()
If Openfd.FileName <> “” Then
    strTexto = My.Computer.FileSystem.ReadAllText(Openfd.FileName,
System.Text.Encoding.Default)
    stLabel.Text = Openfd.FileName.ToString
    txtArchivo.Text = strTexto
End If
```

La variable **strTexto** es de tipo **string**. El evento **Openfd.ShowDialog()** nos permite mostrar el cuadro de diálogo **Abrir**, a través del cual seleccionamos el archivo de



CODIFICACIÓN DE TEXTOS

System.Text.Encoding agrupa las codificaciones que puede contener un archivo. Entre ellas están **ASCII**, **UTF-16**, **ANSI**, **Default**, **UTF-7**, **UTF-8**, **UTF-32** y **Unicode**. Si conocemos la codificación de caracteres que utilizará nuestra aplicación, podremos seleccionarla. Si no, elegimos **Default**, la codificación definida por defecto en el sistema operativo.

texto. A continuación, verificamos que la propiedad **Openfd.FileName** no esté vacía. Ejecutamos la clase **My.Computer.FileSystem.ReadAllText()**, a la cual le pasamos dos parámetros: **Openfd.Filename**, que contiene el nombre de archivo; y **System.Text.Encoding.Default**, para abrir el archivo seleccionado. La propiedad **FileName** proporciona la ruta de acceso completo al archivo mostrada en el pie del Form. **ReadAllText** nos permite leer todo el contenido del archivo de texto para en una primera instancia almacenarlo en la variable **strTexto**.

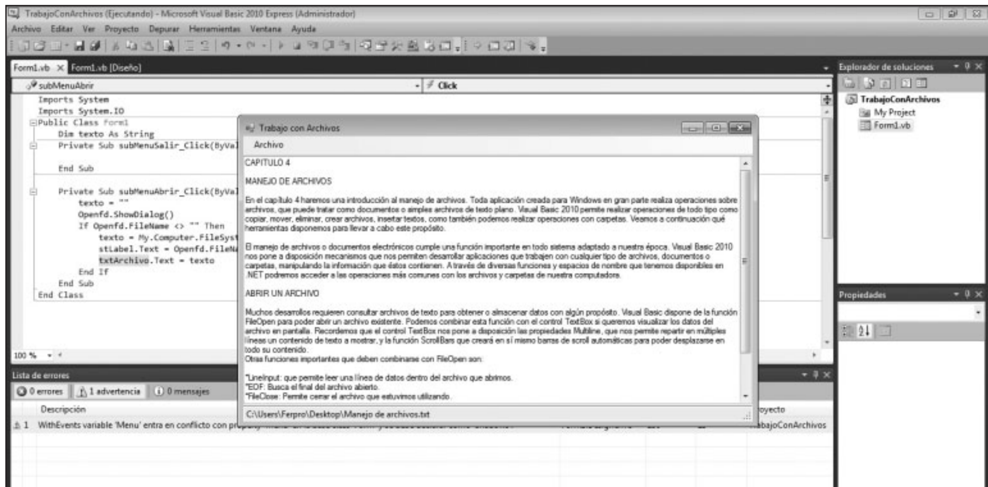


Figura 2. *El resultado del código escrito hasta ahora es un simple editor similar al Bloc de notas, que por el momento nos permite abrir archivos de texto.*

Terminada la apertura y carga del texto, nos queda cerrar el archivo y liberar a **txtArchivo** de todo contenido. A continuación del submenú **Abrir** creamos otro submenú con las propiedades **Name = subMenuCerrar**, **Text = Cerrar**, **ShortCutKeys = CTRL + F4**, y en su evento **Click()** escribimos el siguiente código:

```
strTexto = “”
stLabel.Text = “Listo.”
txtArchivo.Text = strTexto
```

Escribir datos en un archivo

Hemos logrado abrir un archivo de texto y visualizar su contenido. Si no deseamos que sea modificado (es decir, queremos que sea de solo lectura), debemos establecer la propiedad de **txtArchivo ReadOnly = True**. Esto impedirá que el contenido sea alterado. Si queremos que el contenido pueda cambiarse, dejando la propiedad **ReadOnly = False**, será posible hacerlo, pero para que se vea reflejado en el archivo, debemos recurrir a la clase **WriteAllText()**, que contiene parámetros adicionales.

- **Filename**: indica en qué archivo debe escribir el texto.
- **Text**: indica qué texto debe guardar.
- **Append**: valor **Boolean** que indica si el texto por guardar debe agregarse al existente o hay que reemplazar lo que existe.

Al menú de nuestro proyecto, vamos a agregarle un **subItem** más, denominado **Guardar**, cuya propiedad **Name** sea **subMenuGuardar**. En él incluiremos el código para que ejecute el almacenamiento correspondiente. También vamos a asignarle un **ShortCutKey**, que puede ser **CTRL + G**, para tener una función rápida de guardado. En el evento **click()** del submenú **Guardar** escribimos el siguiente código:

```
My.Computer.FileSystem.WriteAllText(stLabel.Text, txtArchivo.Text, False)
```

Ejecutamos la aplicación, volvemos a abrir el archivo y agregamos un texto en él. Luego, presionamos **CTRL + G** para ver si logramos guardar el texto. Para verificarlo, cerramos el archivo desde el menú **Cerrar** y lo abrimos otra vez.

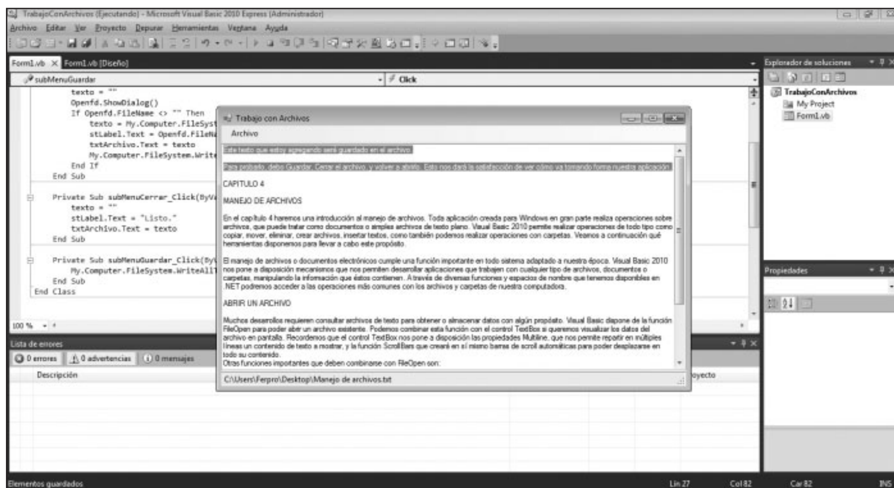


***Figura 3.** Poco a poco, nuestro editor de textos va tomando forma gracias a sus funciones básicas.*

Búsqueda de texto dentro de un archivo

Muchas veces, cuando los textos son extensos, suele suceder que debemos rastrear rápidamente determinada oración pero solo recordamos alguna palabra clave que puede ayudarnos a encontrarla. Para ubicar este texto, necesitamos agregar a nuestro proyecto la opción de buscar. Entonces, creamos un nuevo menú llamado **Edición**, y un submenú que tenga como nombre **Buscar cadena de texto**. El **name** de dicho submenú será **subMenuBuscarTexto**, y su **ShortCutKey**, **CTRL + B**.

Para volver funcional la búsqueda en nuestro editor de texto, en el evento **Click()** de **subMenuBuscarTexto** ingresamos el siguiente código:

```
Dim i As Long, j As Long, strCadena As String, strTexto As String
strCadena = InputBox("Ingrese la cadena de texto a buscar:", "Búsqueda de
texto", strCadena)
If strCadena = "" Then Exit Sub
    i = 1
    j = Len(strCadena)
    strTexto = Trim(txtArchivo.Text)
    For i = 0 To txtArchivo.TextLength
        If Mid(strTexto, i, j) = Trim(strCadena) Then
            txtArchivo.SelectionStart = i
            txtArchivo.SelectionLength = Len(strCadena)
        End If
    Next
```

En él definimos cuatro variables, dos de tipo **Long** (**i** y **j**) y dos de tipo **String** (**strCadena** y **strTexto**). Ahora nos resta invocar a la función **Inputbox** para que se muestre un cuadro de diálogo donde ingresaremos parte del texto que queremos localizar. En la variable **strCadena** guardamos el texto por buscar, y en **strTexto**, el contenido que visualizamos en **txtArchivo**.
Con la sentencia **For** recorremos desde el carácter **1** hasta el final de los caracteres que componen el texto, seleccionando porciones según la longitud que contiene nuestra cadena de búsqueda. Si encontramos mediante comparación igual el texto, las propiedades **SelectionStart** y **SelectionLength** nos permitirán seleccionar la porción coincidente del parámetro de búsqueda ingresado.
Ejecutemos nuestro proyecto y probemos qué tal funciona la búsqueda de texto, abriendo previamente un archivo de texto de nuestra computadora.
El sistema de búsqueda resultante es básico. Con un poco de paciencia e ideas lograremos mejorar el mismo con solo unos pequeños cambios.



COMPLETAR LA APLICACIÓN

Hasta aquí tenemos casi un editor de textos completo, solo nos falta agregar la opción de crear un nuevo archivo. Para hacerlo, utilizamos el control **SaveFileDialog** para ingresar el nombre de archivo con el cual queremos guardar. Por último, repetimos la función de grabación de texto utilizada en este proyecto. ¡Hay que animarse a terminarlo!

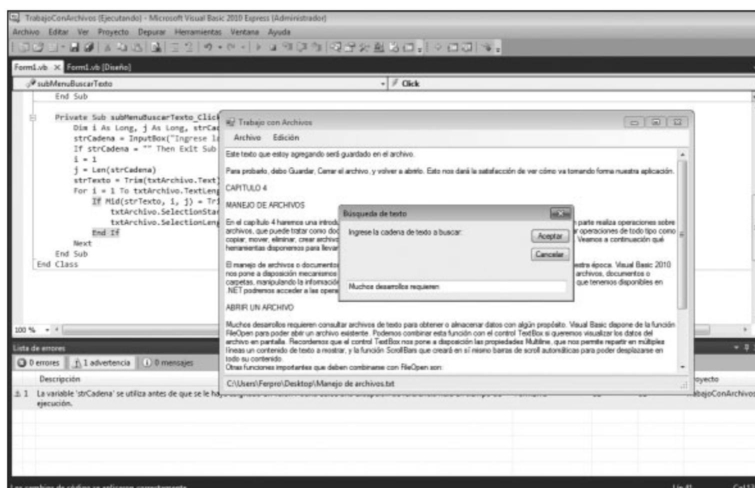


***Figura 4.** Sistema de búsqueda básico pero funcional. Con pocas modificaciones, podremos buscar, en el resto del archivo, más texto coincidente con el ingresado.*

ARCHIVOS DE TEXTO ENRIQUECIDO

Hasta ahora hemos manipulado archivos de texto plano, del tipo **.TXT**. En Visual Basic 2010 también podemos trabajar con archivos de formato enriquecido, similar a los empleados por aplicaciones como **Microsoft Word**, **WordPad** u **OpenOffice Writer** (ver figura que se encuentra a continuación).

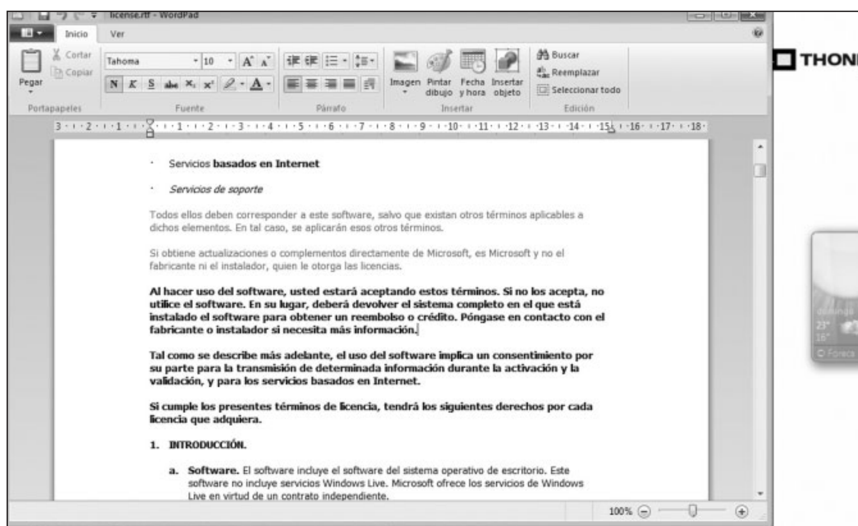


***Figura 5. WordPad** es el procesador de textos que viene integrado en Windows, y que nos permite leer y crear archivos .RTF.*

El control RichTextBox

Visual Basic 2010 incluye **RichTextBox** en su lista de controles. Este control posee las funcionalidades de un **textBox** común, con el plus de que nos permite enriquecer el contenido de su texto, usando **negrita**, **cursiva**, **subrayado** e **hipervínculos**, entre otras funciones. También podemos abrir, escribir y grabar contenido en archivos con extensión **.RTF** (*Rich Text Format*) similar a un documento de Word.
Junto con la revisión de las funciones de este control, aprovecharemos para conocer otros controles que nos permitirán potenciar nuestro proyecto anterior de editor de textos. Iniciemos, entonces, un nuevo proyecto **Aplicación Windows Forms** al que llamaremos **EditorEnriquecido**. En la **Tabla 2** detallamos los controles que serán agregados a este nuevo Editor de textos avanzado.

CONTROL	NOMBRE	FUNCIÓN
ToolStripContainer	tsContainer	Panel contenedor que distribuirá los controles MenuStrip, ToolStrip, StatusStrip y RichTextBox.
MenuStrip	Menus	Menús de nuestra app (Archivo, Edición, Ver).
ToolStrip	tBar	Barra de herramientas.
StatusStrip	sBar	Barra de estado.
RichTextBox	controlRTF	Caja de texto enriquecido que mostrará y permitirá editar archivos de texto y formato RTF.

***Tabla 2.** Controles visibles que tendrá nuestro Editor.*

En primer lugar, debemos agregar **ToolStripContainer**, este control invisible en tiempo de ejecución permitirá que organicemos el resto de los controles.
MenuStrip nos permitirá la creación de tres menús: **Archivo**, **Edición** y **Ver**. Cada uno de ellos tendrá una serie de submenús.

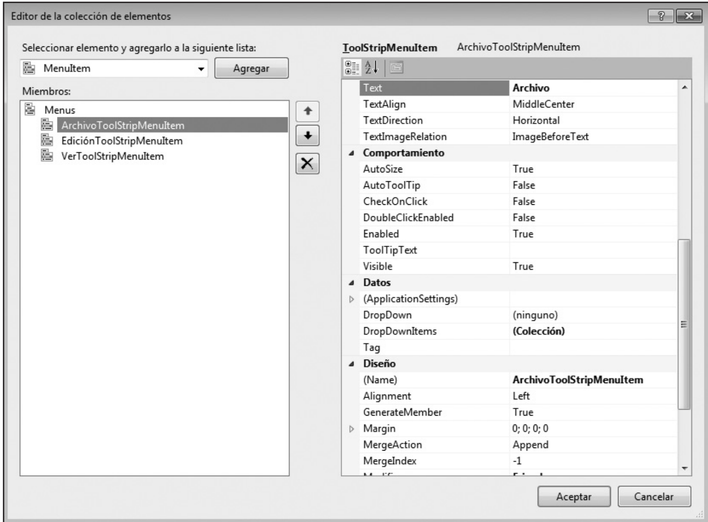


***Figura 6.** Desde el **Editor de la colección de elementos** podemos nombrar y crear los atajos para cada uno de nuestros menús.*

En la **Tabla 3** podemos ver el nombre y acceso directo de cada submenú que compondrá el menú de nuestra aplicación.

NOMBRE MENÚ ARCHIVO	DESCRIPCIÓN	ACCESO RÁPIDO	PROPIEDAD ENABLED
NuevoToolStripMenuItem	Nuevo	CTRL+N	True
AbrirToolStripMenuItem	Abrir	CTRL+A	True
GuardarToolStripMenuItem	Guardar	CTRL+G	False
GuardarComoToolStripMenuItem	Guardar como...	CTRL+SHIFT+G	False
ToolStripMenuItem1	Separador		
CerrarArchivoToolStripMenuItem	Cerrar	CTRL+F4	False
ToolStripMenuItem2	Separador		
SalirToolStripMenuItem	Salir		True
Menú Edición			
DeshacerToolStripMenuItem	Deshacer	CTRL+Z	False
ToolStripMenuItem3	Separador		
CortarToolStripMenuItem	Cortar	CTRL+X	False
CopiarToolStripMenuItem	Copiar	CTRL+C	False
PegarToolStripMenuItem	Pegar	CTRL+V	False
ToolStripMenuItem4	Separador		
SeleccionarTodoToolStripMenuItem	Seleccionar todo	CTRL+E	False
	Buscar	CTRL+B	False
ToolStripMenuItem5	Separador		
InsertarImagenDesdeArchivoToolStripMenuItem	Insertar imagen desde archivo		False
Menú Ver			
ZoomAumentarToolStripMenuItem	Zoom Aumentar		False
ZoomAlejarToolStripMenuItem	Zoom Alejar		False
ToolStripMenuItem6	Separador		
AlinearALaIzquierdaToolStripMenuItem	Alinear a la izquierda		False
AlinearAlCentroToolStripMenuItem	Alinear al centro		False
AlinearALaDerechaToolStripMenuItem	Alinear a la derecha		False
ToolStripMenuItem7	Separador		
CambiarColorDeTextoToolStripMenuItem	Cambiar color de texto		False
CambiarColorDeFondoToolStripMenuItem	Cambiar color de fondo		False
ToolStripMenuItem8	Separador		
CambiarFuenteToolStripMenuItem	Cambiar fuente	CTRL+F	False

***Tabla 3.** Así quedará construido el menú de nuestro Editor de textos .RTF.*

Si bien en las nuevas versiones de Office y Windows, las aplicaciones disponen del menú Ribbon, aún no existe esta característica en Visual Studio.

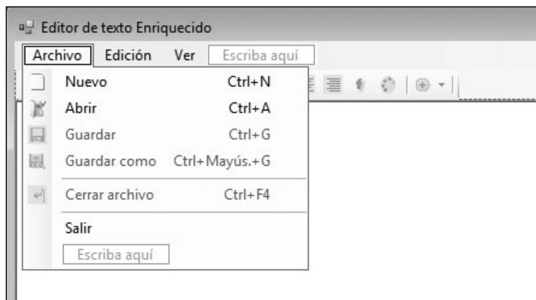


***Figura 7.** Aquí vemos el menú Archivo desplegado en modo de diseño, con sus respectivos submenús habilitados y deshabilitados.*

El siguiente paso es crear la barra de herramientas, el acceso directo a las funciones más comunes. Para hacerlo, arrastramos **ToolStrip** hasta el panel superior de **ToolStripContainer**, justo debajo del menú. **ToolStrip1** será llamado **tBar**.
Para esto debemos conseguir los iconos que utilizaremos como imagen distintiva de nuestra **Toolbar** en cada uno de sus botones. Una vez que los tenemos, los agregamos como recursos en nuestro proyecto. La barra de herramientas quedará compuesta como lo indica la **Tabla 4**, que se encuentra a continuación.

BOTÓN	TOOLTIPTEXT	PROPIEDAD ENABLED
tsBtnNuevo	Nuevo documento	True
tsBtnAbrir	Abrir documento	True
tsBtnGuardar	Guardar documento	False
tsBtnGuardarComo	Guardar documento como...	False
tsBtnCerrar	Cerrar documento	False
ToolStripSeparator1		
tsBtnCortar	Cortar	False
TsBtnCopiar	Copiar	False
TsBtnPegar	Pegar	False
ToolStripSeparator2		
TsBtnIzquierda	Alinear a la izquierda	False
tsBtnCentrar	Alinear al centro	False



INFORMACIÓN SOBRE EL FORMATO RTF

El formato **RTF** fue creado por el equipo de Microsoft Word en 1987 con el objetivo de intercambiar documentos multiplataforma. En un principio, fue implementado en la versión 3.0 de Microsoft Word para Macintosh. Hoy en día, casi todos los procesadores de textos que existen, tanto pagos como gratuitos, son capaces de escribir el formato **RTF.1**.

BOTÓN	TOOLTIPTEXT	PROPIEDAD ENABLED
TsBtnDerecha	Alinear a la derecha	False
TsBtnFonts	Seleccionar fuente	False
TsBtnColor	Seleccionar color de fuente	False
ToolStripSeparator3		
tsBtnZoom – Zoom (ToolStripSplitButton – permite crear un menú desplegable de botones)		False
ZoomAcercarToolStripMenuItem	Zoom Acercar (+)	True
ZoomAlejarToolStripMenuItem	Zoom Alejar (+)	True
ToolStripMenuItemSeparador		
VistaNormal11ToolStripMenuItem	Vista normal (1:1)	True
ToolStripSeparator4		

***Tabla 4.** Cada uno de los botones de la barra de herramientas, con su respectivo **ToolTipText**, que informará sobre su función al usuario.*

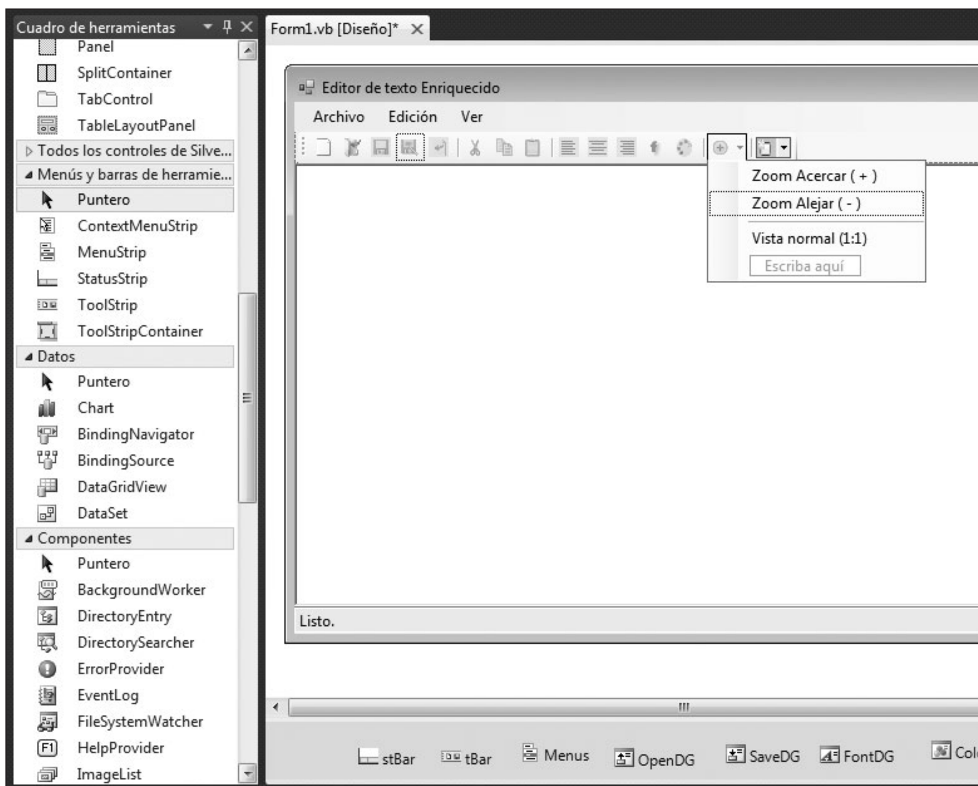


***Figura 8.** Nuestra barra de herramientas terminada, con todos los estados **Enabled** y **Disabled** de sus botones.*

Hasta ahora, hemos desarrollado el menú principal y la barra de herramientas para nuestro editor de texto enriquecido. A continuación, vamos a agregar el control **RichTextBox**, donde cargaremos y editaremos los archivos **.RTF** y **TXT**.

Arrastramos el control desde el Cuadro de herramientas al panel centrar de **ToolStripContainer**. El control tendrá las propiedades detalladas en la **Tabla 5**.

PROPIEDAD	VALOR
Name	controlRTF
ReadOnly	True
Multiline	True
AcceptTabs	True
BackColor	White
DetectURLs	True
Font	Calibri; 9.75pt
Scrollbars	ForcedBoth
ZoomFactor	1

***Tabla 5.** Estas son las principales propiedades del control RichTextBox. DetectURLs=True permitirá establecer el formato determinado de una URL dentro del texto.*



***Figura 9.** Marcar la opción Acoplar en contenedor primario permitirá ajustar el control al ancho de su contenedor automáticamente.*

El último de los controles por agregar es **StatusStrip**. Lo añadimos en el panel inferior central de **ToolstripContainer**, y ajustamos su propiedad **Name** como **stBar**. Luego, agregamos dos **ToolStripStatusLabel**, como indica la **Tabla 6**.

NOMBRE	VALOR
stLabel1	
Text	Listo
TextAlign	MiddleLeft
AutoSize	True
stLabelCaracteres	
Text	Caracteres: 0
TextAlign	MiddleLeft
AutoSize	False

***Tabla 6.** Los dos elementos que tendrá la barra de tareas de nuestro Editor de textos. En el primero, se informará el nombre de archivo en uso, y en el segundo, la cantidad de caracteres escritos.*

Al igual que toda aplicación Windows, nuestro desarrollo dispondrá de las características más comunes: Menús, Barra de herramientas y Barra de estado.

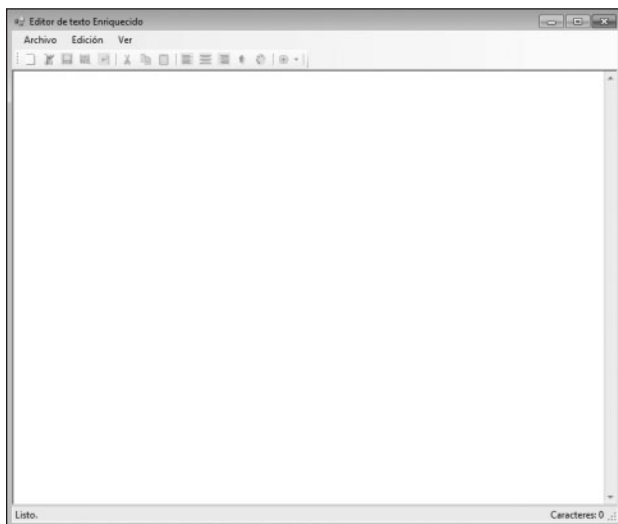


Figura 10. *Si bien hasta el momento no tiene funcionalidad, podemos ejecutar nuestro Editor de textos para ver cómo ha quedado su diseño.*

Desde aquí comenzaremos a darle funcionalidad a nuestra aplicación. Dado que los menús y la barra de herramientas comparten funciones que el programa tendrá, debemos tratar de no repetir el código de este por dos motivos: primero, para optimizar nuestra aplicación, y segundo, para que cuando queramos hacer alguna corrección sobre el código de un menú, lo hagamos en un solo lugar y este quede funcional tanto en el menú como en el botón correspondiente en la barra de herramientas. Para este fin crearemos procedimientos dentro de nuestra aplicación, que luego serán llamados desde el menú y desde la barra de herramientas.
Lo primero que haremos será, a continuación de la clase **FrmEditor**, declarar unas variables que nos resultarán de utilidad dentro de nuestro proyecto, para lo cual escribimos, en el código fuente, lo siguiente:

```
Public Class frmEditor
     Dim pathFileName As String
     Dim bCambio As Boolean
     Dim zoom As Single
…
```

- **pathFileName** manejará el archivo abierto o guardado que trabajaremos desde el Editor. Esto nos agilizará las funciones de los cuadros de diálogo **Abrir** y **Guardar**, como también la función de la barra de estado.
- **bCambio** es una variable **boolean** que nos indicará cuando modifiquemos el documento que estamos utilizando. Esto nos dirá si, al intentar cerrarlo, la aplicación

debe guardar de manera automática el archivo actualmente en edición, o si tiene que ofrecernos que lo hagamos nosotros.

- **Zoom** contendrá el valor de aumento que aplicaremos en nuestro editor para acercar o alejar el documento en el cual nos encontramos trabajando.

El segundo procedimiento que crearemos se llama **CambioControles**, y se encargará de administrar el estado Enabled de cada botón, control y menú de la aplicación:

```
Public Sub CambioControles()
tsBtnNuevo.Enabled = Not tsBtnNuevo.Enabled
tsBtnAbrir.Enabled = Not tsBtnAbrir.Enabled
tsBtnGuardar.Enabled = Not tsBtnGuardar.Enabled
tsBtnGuardarComo.Enabled = Not tsBtnGuardarComo.Enabled
tsBtnCerrar.Enabled = Not tsBtnCerrar.Enabled
tsBtnCortar.Enabled = Not tsBtnCortar.Enabled
tsBtnCopiar.Enabled = Not tsBtnCopiar.Enabled
tsBtnPegar.Enabled = Not tsBtnPegar.Enabled
tsBtnIzquierda.Enabled = Not tsBtnIzquierda.Enabled
tsBtnCentrar.Enabled = Not tsBtnCentrar.Enabled
tsBtnDerecha.Enabled = Not tsBtnDerecha.Enabled
tsBtnFonts.Enabled = Not tsBtnFonts.Enabled
tsBtnColor.Enabled = Not tsBtnColor.Enabled
tsBtnZoom.Enabled = Not tsBtnZoom.Enabled
NuevoToolStripMenuItem.Enabled = Not NuevoToolStripMenuItem.Enabled
AbrirToolStripMenuItem.Enabled = Not AbrirToolStripMenuItem.Enabled
GuardarToolStripMenuItem.Enabled = Not GuardarToolStripMenuItem.Enabled
GuardarComoToolStripMenuItem.Enabled = Not
GuardarComoToolStripMenuItem.Enabled
CerrarArchivoToolStripMenuItem.Enabled = Not CerrarArchivoToolStripMe
nuItem.Enabled
SalirToolStripMenuItem.Enabled = Not SalirToolStripMenuItem.Enabled
DeshacerToolStripMenuItem.Enabled = Not DeshacerToolStripMenuItem.Enabled
CortarToolStripMenuItem.Enabled = Not CortarToolStripMenuItem.Enabled
CopiarToolStripMenuItem.Enabled = Not CopiarToolStripMenuItem.Enabled
PegarToolStripMenuItem.Enabled = Not PegarToolStripMenuItem.Enabled
SeleccionarTodoToolStripMenuItem.Enabled = Not SeleccionarTodoToolStripMe
nuItem.Enabled
BuscarToolStripMenuItem.Enabled = Not BuscarToolStripMenuItem.Enabled
InsertarImagenDesdeArchivoToolStripMenuItem.Enabled = Not InsertarImagenDes
deArchivoToolStripMenuItem.Enabled
```

```
ZoomAumentarToolStripMenuItem.Enabled = Not
ZoomAumentarToolStripMenuItem.Enabled
ZoomAlejarToolStripMenuItem.Enabled = Not ZoomAlejarToolStripMenuItem.Enabled
AlinearALaIzquierdaToolStripMenuItem.Enabled = Not AlinearALaIzquierdaToolS
tripMenuItem.Enabled
AlinearALaDerechaToolStripMenuItem.Enabled = Not AlinearALaDerechaToolStripMe
nuItem.Enabled
AlinearAlCentroToolStripMenuItem.Enabled = Not AlinearAlCentroToolStripMe
nuItem.Enabled
CambiarFuenteToolStripMenuItem.Enabled = Not
CambiarFuenteToolStripMenuItem.Enabled
End Sub
```

A continuación, agregaremos el código correspondiente a los procedimientos para acercar y alejar el documento a través del zoom, y el recuento de caracteres para saber cuánto hemos escrito en la aplicación:
Poco a poco, nuestro proyecto comienza a volverse más funcional.

```
'Aumentar el Zoom

Private Sub AumentoZoom()
If controlRTF.TextLength > 0 Then
    If zoom < 7 And zoom > 0.49 Then zoom = zoom + 0.5
    controlRTF.ZoomFactor = zoom
End If
End Sub

'Disminuir el Zoom

Private Sub DisminuyoZoom()
```



MANEJO DE LA PROPIEDAD ZOOMFACTOR

La propiedad **ZoomFactor** tiene un valor mínimo de 0.5 y un máximo de 63.99. En nuestro Editor de textos enriquecido manejamos los valores 0.5 como mínimo y 7 como máximo, con una escala de 0.5 para aplicar el zoom. Si lo deseamos, podemos alterar el resultado aquí establecido para mejorar su precisión aprovechando los valores máximos y mínimos.

```
If controlRTF.TextLength > 0 Then
     If zoom < 7 And zoom > 0.5 Then zoom = zoom - 0.5
     controlRTF.ZoomFactor = zoom
End If
End Sub


'Recuento de caracteres de nuestro documento

Sub cuentoCaracteres()
     Dim T As Long = Trim(controlRTF.TextLength)
     stLabelCaracteres.Text = "Caracteres: " & T
End Sub
```

El evento **Load** de **frmEditor** contendrá el siguiente código:

```
Private Sub frmEditor_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
     bCambio = False
     zoom = 1
End Sub
```

En él ajustamos que la variable cambio sea **False** y el valor de **Zoom** sea 1.
El control **RichTextBox** contiene un evento llamado **TextChanged**, que detecta todo el cambio que se haga sobre el documento en uso. Allí situaremos el procedimiento **cuentoCaracteres()** y cambiaremos el valor de **bCambio** a **True**:

```
Private Sub controlRTF_TextChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles controlRTF.TextChanged
     cuentoCaracteres()
     bCambio = True
End Sub
```

RichTextBox posee una propiedad denominada **SelectionAlignment**, a través de la cual podemos seleccionar una porción de texto o el contenido total del documento y establecer si estará alineada a la **izquierda**, **derecha** o al **centro** del documento. Con este fin creamos los siguientes procedimientos:
Lamentablemente el control **RichTextBox** no cuenta con la opción **Justificado**.

```
Private Sub alinearIzquierda()
    controlRTF.SelectionAlignment = HorizontalAlignment.Left
End Sub
Private Sub alinearCentro()
    controlRTF.SelectionAlignment = HorizontalAlignment.Center
End Sub
Private Sub alinearDerecha()
    controlRTF.SelectionAlignment = HorizontalAlignment.Right
End Sub
```

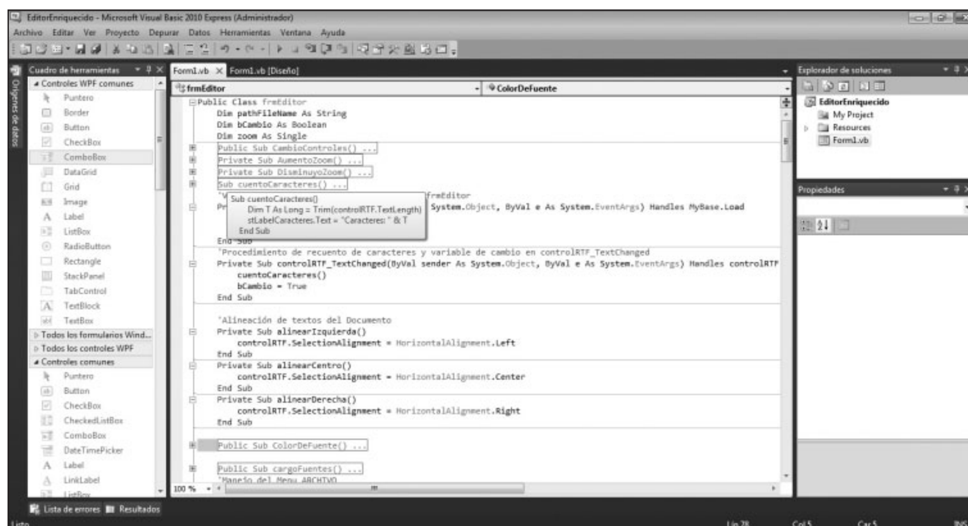


Figura 11. *Para ordenar nuestras funciones y procedimientos podemos utilizar la función [+] al lado de cada procedimiento y evento. Esto permitirá cerrarlo y, así, limpiar el área de trabajo.*

CONTROLES AVANZADOS: CUADROS DE DIÁLOGO

Para abrir y guardar documentos, como así también para establecer fuentes, tamaño de texto, subrayado, estilos y colores, debemos recurrir a los cuadros de diálogo **OpenFileDialog**, **SaveFileDialog**, **FontDialog** y **ColorDialog**, comunes a la mayoría de las aplicaciones. Estos conducen de manera directa a las carpetas y archivos de nuestro disco, y también a las propiedades de las fuentes y colores del sistema. De este modo, tenemos la ventaja de que no necesitamos programar la aplicación creando cada uno de ellos en forma manual: con solo ajustar algunas propiedades, podemos disponer de ellos en nuestro proyecto.

OpenFileDialog

Este cuadro de diálogo permitirá abrir los archivos desde el disco. Lo arrastramos desde el **Cuadro de herramientas** y ajustamos la propiedad **Name = OpenDG**.

SaveFileDialog

Este otro cuadro de diálogo nos permitirá guardar nuestros documentos y prever ciertos cambios al momento de hacerlo, como si existe el directorio donde guardaremos el archivo, y agregar la extensión correspondiente si es que no la incluimos en el nombre. Agregamos este control con su propiedad **Name = SaveDG**.

ColorDialog

Su función es establecer el color del texto y del fondo de página de nuestros documentos. Su paleta de colores nos permitirá seleccionar entre los tonos más comunes, o buscar dentro de la gama de colores personalizados. Agregamos este control y establecemos su propiedad **Name = ColorDG**:

```
Public Sub ColorDeFuente()
    ColorDG.AllowFullOpen = True
    ColorDG.Color = controlRTF.SelectionColor
    ColorDG.ShowDialog()
    If Windows.Forms.DialogResult.OK Then
        controlRTF.SelectionColor = ColorDG.Color
    End If
End Sub
```

En el procedimiento de uso de selección de color tenemos la propiedad **Color**, que nos indica el seleccionado. Con **ShowDialog()** mostramos el cuadro de diálogo. Al igual que en los casos anteriores, mediante **DialogResult.OK** obtenemos si se presionó el botón **Aceptar**, y según esta respuesta, aplicamos el color elegido sobre el texto seleccionado del control **RichTextBox**. Incluso podremos extender la gama básica de colores con la opción Colores Personalizados.



RECUENTO DE PALABRAS

Así como desarrollamos un contador de caracteres tipiados, podemos hacer lo mismo contando las palabras que contiene un documento. La opción para lograrlo es utilizar la búsqueda implementada en el menú **subMenuBuscarTexto**, buscando un espacio vacio **" "**, y combinándola con una recorrida por todo el documento, o combinándola con la función **Split()**.

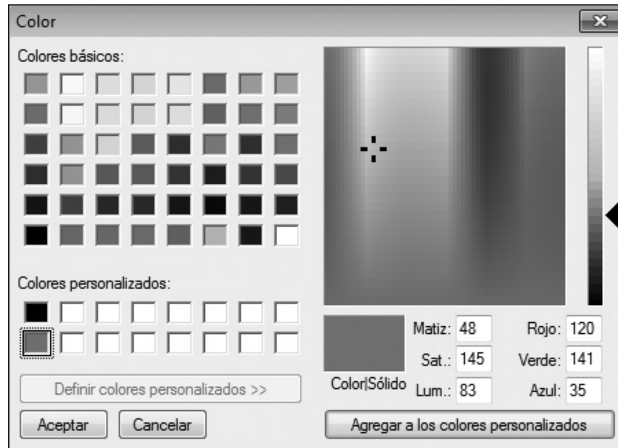


Figura 12. *La paleta de color permitirá que apliquemos el color deseado sobre el texto de nuestro documento.*

FontDialog

FontDialog, al igual que los anteriores cuadros de diálogo, permite establecer la fuente sobre una porción de texto, su tamaño, el estilo **Negrita**, **Cursiva**, o ambos, y también si el texto seleccionado debe estar **subrayado** o **tachado**. Agregamos este control y establecemos su propiedad **Name = FontDG**:

```
Public Sub cargoFuentes()
    FontDG.Font = controlRTF.SelectionFont
    FontDG.ShowDialog()
    If Windows.Forms.DialogResult.OK Then
        controlRTF.SelectionFont = FontDG.Font
    End If
End Sub
```

La propiedad **Font** permite establecer o especificar la fuente que vamos a utilizar. A través de **ShowDialog()** podemos visualizar el cuadro de diálogo de fuentes.



CUADRO DE DIÁLOGO IMPRESIÓN

Microsoft no incluyó la posibilidad de imprimir documentos **.RTF** mediante los cuadros de diálogo **Impresión**, pero de todos modos, sí se puede imprimirlos. En la dirección **http://support.microsoft.com/kb/811401** encontraremos el soporte oficial de la empresa que nos explica de manera detallada cómo llevar a cabo esta función.

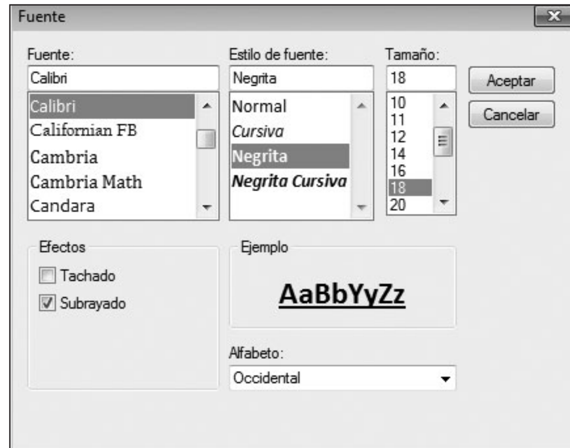


Figura 13. *El cuadro de diálogo de* ***Fuentes****, en acción.*



Figura 14. *Los controles Cuadros de diálogo no pueden visualizarse en modo Diseño. Sólo se verán en tiempo de ejecución de la aplicación, y serán invocados mediante código.*

Para que los cuadros de diálogo se vuelvan funcionales, vamos a agregar a continuación el código de cada uno de ellos, debajo de los procedimientos creados anteriormente:

```
Public Sub AbrirDocumento()
   If Trim(Me.controlRTF.Text) = "" Then
    With OpenDG
        .Title = "Abrir documento"
        .FileName = ""
        .FilterIndex = 0
        .Filter = "Archivos RTF | *.rtf|Archivos de texto | *.txt"
        .CheckFileExists = True
        .ShowDialog()
    End With
        If Windows.Forms.DialogResult.OK Then
        If OpenDG.FileName <> "" Then                      controlRTF.Load
File(OpenDG.FileName)
    pathFileName = OpenDG.FileName
    controlRTF.ReadOnly = False
        If controlRTF.Visible = False Then controlRTF.Visible = True
        End If
```

```
        End If
    End If
End Sub
```

A través de **With <nombredelcontrol>**, podemos acceder a varias propiedades juntas del control **OpenDG**, sin necesidad de repetir su nombre por cada una de ellas. Al finalizar, terminamos la instrucción con **End With**.

La propiedad **Title** permite establecer el nombre de la ventana; en este caso, será **"Abrir archivo"**. La propiedad **Filter** permite establecer los tipos de archivo que se mostrarán en el cuadro de diálogo. **CheckFileExists = True** permite asegurarnos de que no se produzca una excepción si es que escribimos directamente un nombre de archivo que no existe o no se encuentra en la carpeta en la que estamos posicionados. **ShowDialog()** es el evento que mostrará el cuadro de diálogo **Abrir**. Si aceptamos este cuadro de diálogo, se producirá un evento **DialogResult.OK**, lo que indica que presionamos **Aceptar** u **OK**, dependiendo del lenguaje de nuestro sistema operativo. El procedimiento siguiente es verificar si con **OK** el cuadro de diálogo contiene un **nombre de archivo** y un **path** válidos. **PathFileName** recibe como valor la ruta completa al documento abierto.

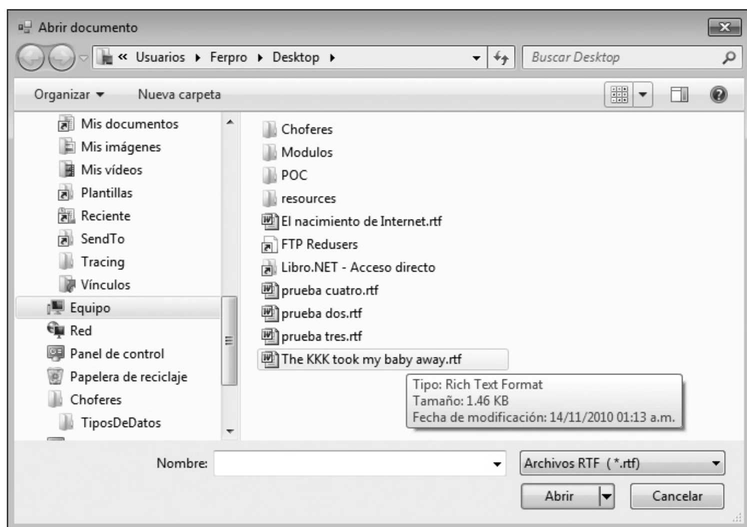


***Figura 15.** Esta es la manera en la que quedará el cuadro de diálogo **Abrir documento**.*

A continuación, escribimos el procedimiento para guardar nuestros documentos:

```
Public Sub GuardarDocumento()
    If pathFileName <> "" Then
```

```
            controlRTF.SaveFile(pathFileName)
            Exit Sub
        Else
            With SaveDG
                .Filter = “Archivos RTF | *.rtf|Archivos de texto | *.txt”
                .FilterIndex = 0
                .CheckPathExists = True
                .ShowDialog()
            End With
        End If
        If Windows.Forms.DialogResult.OK Then
            If SaveDG.FileName <> “” Then
                controlRTF.SaveFile(SaveDG.FileName)
                bCambio = False
            End If
        End If
    End Sub
```

El procedimiento verifica como primera medida si **PathFileName** tiene valor. De ser cierto, directamente ejecuta el método **controlRTF.SaveFile**, indicando la ruta de nuestro documento almacenada en **PathFileName**. Si no tiene valor, se ejecuta el cuadro de diálogo **Guardar documento**, para que podamos elegir la carpeta y el nombre del documento.
La propiedad **Filter**, tal como en el cuadro de diálogo **Abrir documento**, indica los tipos de archivo que se pueden guardar.

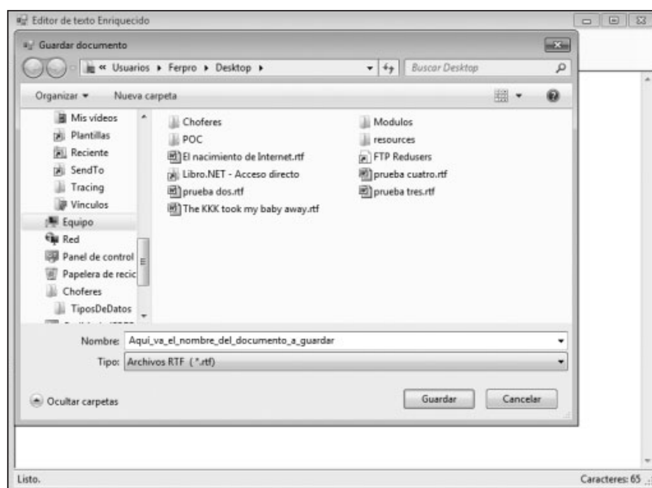


***Figura 16.** El cuadro de diálogo **Guardar documento**, en acción. Como vemos, es similar al cuadro **Abrir**.*

Luego, agregamos el código para que funcione la opción **Guardar documento como...**:

```
Public Sub GuardarDocumentoComo()
    SaveDG.Title = "Guardar documento como"
    GuardarDocumento()
    SaveDG.Title = "Guardar documento"
End Sub
```

Este procedimiento es mucho más abreviado que el anterior, dado que solo se produce un cambio en el cuadro de diálogo **Guardar**: el título. **SaveDG.Title** especifica que cambiará el título, luego se llama al procedimiento **GuardarDocumento()** para hacer uso de sus funciones ya creadas, y una vez procesado esto, se restituye la propiedad **Title** a **Guardar documento**.
Por último, nos queda crear algunos procedimientos más antes de volcarlos dentro de los menús y botones de la aplicación:

```
Public Sub nuevoDocumento()
    If controlRTF.TextLength <> 0 And pathFileName = "" Or bCambio Then
        Dim msg = "¿Desea guardar el Documento actual?"
        If MessageBox.Show(msg, "Guardar documento", MessageBoxButtons.YesNo,
 MessageBoxIcon.Question) = Windows.Forms.DialogResult.Yes Then
            GuardarDocumento()
        End If
    End If
    pathFileName = ""
    controlRTF.ResetText()
    controlRTF.ReadOnly = False
    controlRTF.Visible = True
    controlRTF.Focus()
End Sub
```



DIVERSOS USOS DE RICHTEXTBOX

RichTextBox permite manipular documentos de texto como formato **.RTF**. Muchos programadores encontraron un aliado en este control, para evitar que, en las versiones antiguas de Visual Basic, hubiera que manipular cadenas de texto mediante el uso de **Open "archivo" for input as #1**. **Line Input #1** y **Close #1**, aunque las buenas prácticas no recomendaban hacerlo.

El procedimiento **nuevoDocumento()** revisa en primera instancia si el texto de **controlRTF** no es vacío o si **pathFileName** tiene un valor asignado. Si alguno de los dos es cierto, antes de crear un documento en blanco, consulta al usuario si quiere guardar el documento actual. Dependiendo de la respuesta, ejecutará o no el procedimiento **GuardarDocumento()**. Luego de hacerlo, se limpiarán las variables y los textos para poder trabajar sobre un documento nuevo.
Veamos, a continuación, el código que ejecuta el evento recién comentado.

```
Public Sub CerrarDocumento()
    If pathFileName <> "" Then controlRTF.SaveFile(pathFileName)
    If bCambio And pathFileName = "" Then GuardarDocumento()
    controlRTF.ReadOnly = True
    controlRTF.Text = ""
    pathFileName = ""
    bCambio = False
    CambioControles()
End Sub
```

CerrarDocumento() permite cerrar el documento actual y dejar la aplicación como si recién la hubiésemos iniciado. Se encarga de revisar las variables **pathFileName** y **bCambio** para comprobar si tienen algún valor que indique que trabajamos sobre un documento. De ser así, sugerirá guardarlo. Luego, limpiará la superficie de trabajo y ocultará el control **controlRTF**. Por último, establecerá los valores iniciales de los menús y la barra de herramientas:

```
Public Sub insertarImagen()
    With OpenDG
        .Title = "Insertar imagen"
        .FileName = ""
        .Filter = "Imagen JPG | *.jpg|Imagen Gif | *.gif"
        .ShowDialog()
    End With
    If Windows.Forms.DialogResult.OK Then
        If OpenDG.FileName <> "" Then
           Clipboard.SetImage(Image.FromFile(OpenDG.FileName))
            controlRTF.Paste()
        End If
    End If
End Sub
```

insertarImagen() es otra función, que permitirá que agreguemos a nuestro documento imágenes del tipo **JPG** o **GIF**. El procedimiento es abrir un cuadro de diálogo para seleccionar la imagen que queremos insertar, y luego copiarla al Portapapeles e insertarla en el documento. **Clipboard.SetImage()** permite copiar una imagen al Portapapeles. El método **controlRTF.Paste()** se usa para pegar el contenido que haya en él en nuestro **RichTextBox**.

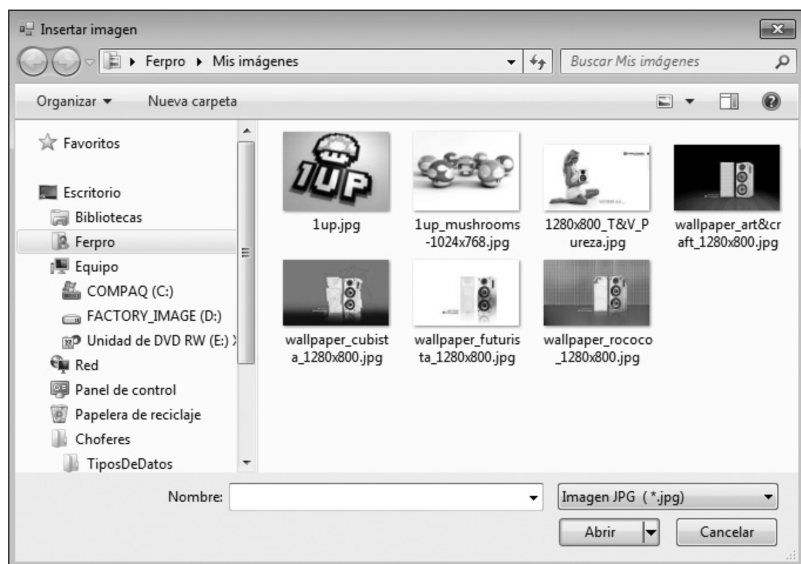


***Figura 17.** El cuadro de diálogo Insertar imagen permitirá que exploremos nuestro disco en busca de un gráfico para el documento.*

Ya hemos creado los procedimientos que evocarán las funciones de nuestro **Editor de texto enriquecido**. Para que quede completamente funcional, solo nos resta llamar a cada procedimiento desde los menús y botones de la barra de herramientas. Copiamos las instrucciones detalladas a continuación dentro de cada evento **Click()** de los controles. Para acotar el extenso texto que esto lleva, se especifica el nombre del procedimiento, el código que contendrá, y se reemplazan los parámetros adicionales del evento **Click()** con tres puntos consecutivos:



MEJORAR EL USO DE CUADROS DE DIÁLOGO

Podemos potenciar al máximo el uso de nuestros cuadros de diálogo **Abrir** y **Guardar**, integrando cada uno de ellos en una función o procedimiento al cual podemos indicarle parámetros adicionales, como **Título**, **Filtro**, **Carpeta inicial** e **índice de filtro**. De esta manera, el código escrito será mucho más reutilizable en todo nuestro proyecto.

```
Private Sub tsBtnFonts_Click(…
    cargoFuentes()
End Sub

Private Sub NuevoToolStripMenuItem_Click(…
    CambioControles()
    nuevoDocumento()
End Sub

Private Sub tsBtnNuevo_Click(…
    CambioControles()
    nuevoDocumento()
End Sub

Private Sub CambiarFuenteToolStripMenuItem_Click(…
    cargoFuentes()
End Sub

Private Sub AbrirToolStripMenuItem_Click…
    CambioControles()
    AbrirDocumento()
End Sub

Private Sub tsBtnAbrir_Click(…
    CambioControles()
    AbrirDocumento()
    controlRTF.Visible = True
    controlRTF.Focus()
End Sub

Private Sub GuardarToolStripMenuItem_Click(…
    GuardarDocumento()
End Sub

Private Sub tsBtnGuardar_Click(…
    GuardarDocumento()
End Sub

Private Sub GuardarComoToolStripMenuItem_Click…
    GuardarDocumentoComo()
```

```
    End Sub

    Private Sub tsBtnGuardarComo_Click(…
        GuardarDocumentoComo()
    End Sub

    Private Sub CambiarColorDeTextoToolStripMenuItem_Click(…
        ColorDeFuente()
    End Sub

    Private Sub tsBtnColor_Click(…
        ColorDeFuente()
    End Sub

    Private Sub CerrarArchivoToolStripMenuItem_Click(…
        CerrarDocumento()
    End Sub

    Private Sub tsBtnCerrar_Click(…
        CerrarDocumento()
    End Sub

    Private Sub tsBtnIzquierda_Click(…
        alinearIzquierda()
    End Sub

    Private Sub ZoomAcercarToolStripMenuItem_Click(… ZoomAcercarToolStrip
MenuItem.Click
        AumentoZoom()
    End Sub

    Private Sub ZoomAlejarToolStripMenuItem1_Click(… ZoomAlejarToolStripMe
nuItem1.Click
        DisminuyoZoom()
    End Sub

    Private Sub AlinearALaIzquierdaToolStripMenuItem_Click(…
        alinearIzquierda()
    End Sub
```

```
Private Sub ZoomAumentarToolStripMenuItem_Click(…
    AumentoZoom()
End Sub

Private Sub ZoomAlejarToolStripMenuItem_Click(…
    DisminuyoZoom()
End Sub

Private Sub VistaNormal11ToolStripMenuItem_Click(…
    zoom = 1
    If controlRTF.TextLength > 0 Then controlRTF.ZoomFactor = zoom
End Sub

Private Sub AlinearAlCentroToolStripMenuItem_Click(…
    alinearCentro()
End Sub

Private Sub AlinearALaDerechaToolStripMenuItem_Click…
    alinearDerecha()
End Sub

Private Sub tsBtnCentrar_Click…
    alinearCentro()
End Sub

Private Sub tsBtnDerecha_Click(…
    alinearDerecha()
End Sub

Private Sub InsertarImagenDesdeArchivoToolStripMenuItem_Click(…
    InsertarImagen()
End Sub

Private Sub CortarToolStripMenuItem_Click(…
    controlRTF.Cut()
End Sub

Private Sub CopiarToolStripMenuItem_Click(…
    controlRTF.Copy()
End Sub
```

```
Private Sub PegarToolStripMenuItem_Click(…
    controlRTF.Paste()
End Sub

Private Sub DeshacerToolStripMenuItem_Click(…
    controlRTF.Undo()
End Sub
```

Con esto, damos por finalizada la programación de nuestro Editor de texto. Guardamos el proyecto y lo ejecutamos para ver el programa en acción.

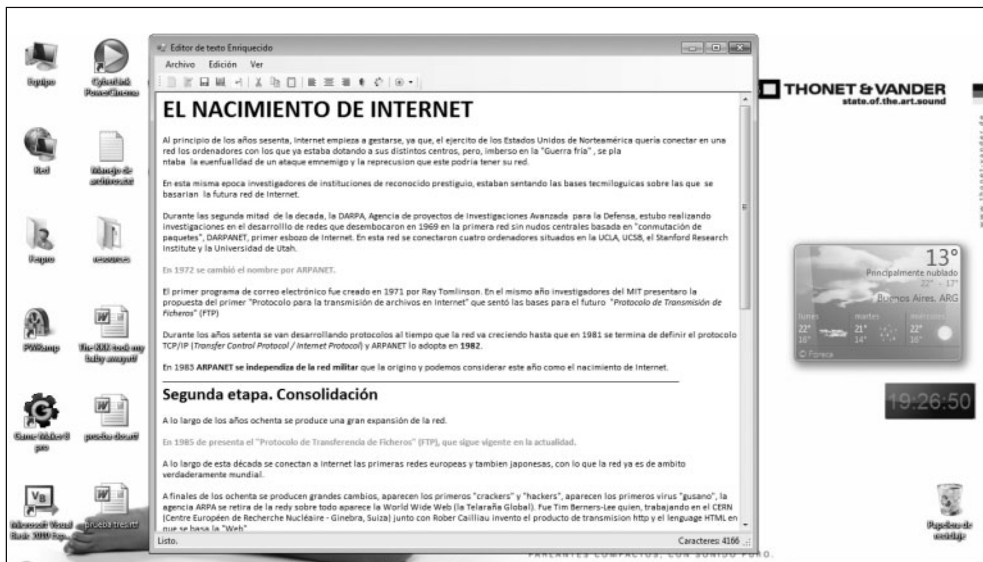


Figura 18. *Nuestro proyecto ya es una realidad. El Editor de texto enriquecido está en funcionamiento.*

Para probar el correcto funcionamiento del proyecto, podemos abrir cualquier documento con extensión **.DOC** desde **Microsoft Word**, guardarlo como documento de texto enriquecido **.RTF** y, luego, abrirlo desde nuestra aplicación.

Otra de las alternativas que tenemos para testear nuestra aplicación es la de crear un documento desde cero, agregar texto en él y comenzar su formateo. No olvidemos probar toda su funcionalidad en cuanto a la inserción de imágenes.

Este proyecto también soporta la posibilidad de copiar una imagen desde Internet a través del navegador, y luego, con la opción **Edición/Pegar**, insertarla en nuestro programa. Esta imagen será guardada localmente en nuestro documento.

Hasta ahora hemos podido comprobar que si bien el control RichTextBox es básico en comparativa con MS Word, aun así posee prestaciones interesantes para explorar.

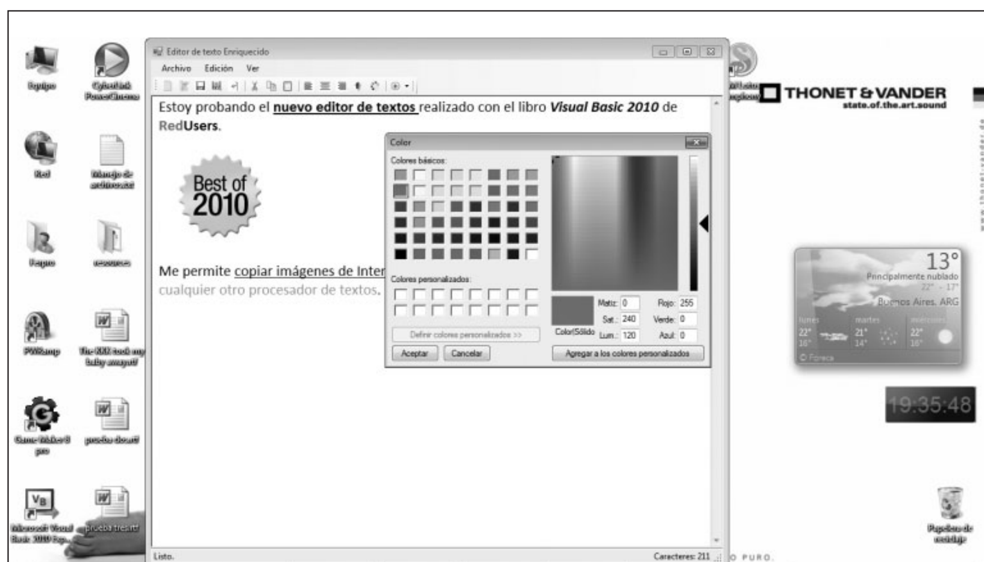


Figura 19. *Como podemos apreciar, insertamos una imagen extraída directamente de Internet a través del Portapapeles de Windows.*

Una vez que testeamos al máximo el editor, podemos grabar el archivo que creamos, cerrar la aplicación y volver a abrirla, para luego abrir el archivo grabado previamente y verificar que su formato siga aplicado.

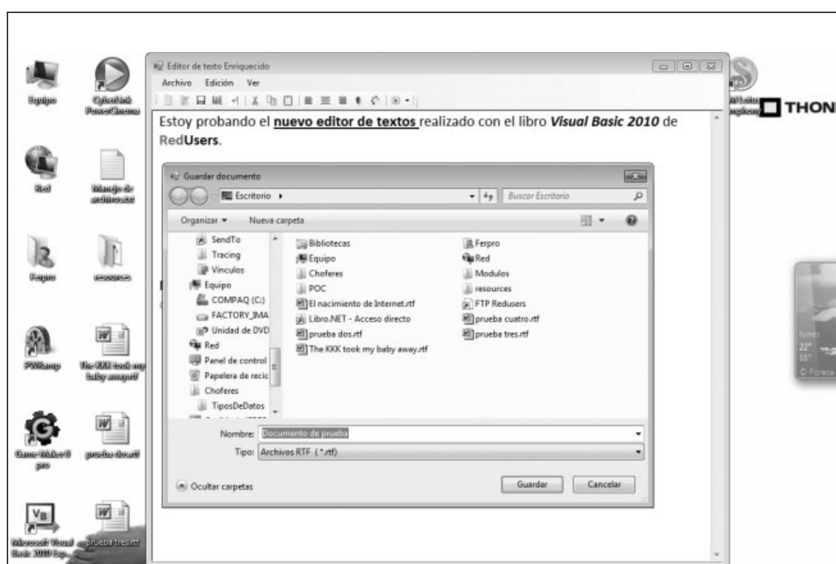


Figura 20. *Esta última prueba permite verificar que el proyecto ha sido todo un éxito.*

Por último, queda probar la función de zoom.



Figura 21. *ZoomFactor()* permite hacer un aumento controlado sobre el documento.

MANEJO DE ARCHIVOS Y CARPETAS

Como mencionamos al principio del capítulo, también es posible manejar archivos y carpetas en Visual Basic 2010. Esto incluye explorar y listar las unidades de la computadora; realizar operaciones de copiar, crear, mover y eliminar archivos y carpetas; y explorar el contenido de un disco.

Unidades de disco

Iniciemos un nuevo proyecto **Application Windows Forms** al cual llamaremos **Archivos y carpetas**. A continuación, vamos a incluir un control **GroupBox** y, dentro de él, un **ListBox**. **GroupBox** es un contenedor que permite agrupar un conjunto de controles para ordenar la interfaz de la aplicación. A continuación, incluiremos un nuevo procedimiento a través del cual listaremos todas las unidades de disco, o drives que contiene nuestra computadora:

```
Public Sub listoUnidades()
    lbDiscos.Items.Clear()
    For Each strDiscos In My.Computer.FileSystem.Drives
        lbDiscos.Items.Add(strDiscos.Name)
    Next
End Sub
```

Invocando el namespace **My.Computer.FileSystem.Drives**, podemos obtener una colección de todas las unidades que hay en nuestra computadora. Dicha colección se almacena en la variable **strDiscos**, a la que recorremos mediante la instrucción **For Each** para agregar un nuevo ítem en el control **lbDiscos**.
Acto seguido, llamamos al procedimiento **listoUnidades()** en el evento **Form_Load**. Ejecutamos nuestro proyecto para ver el resultado.

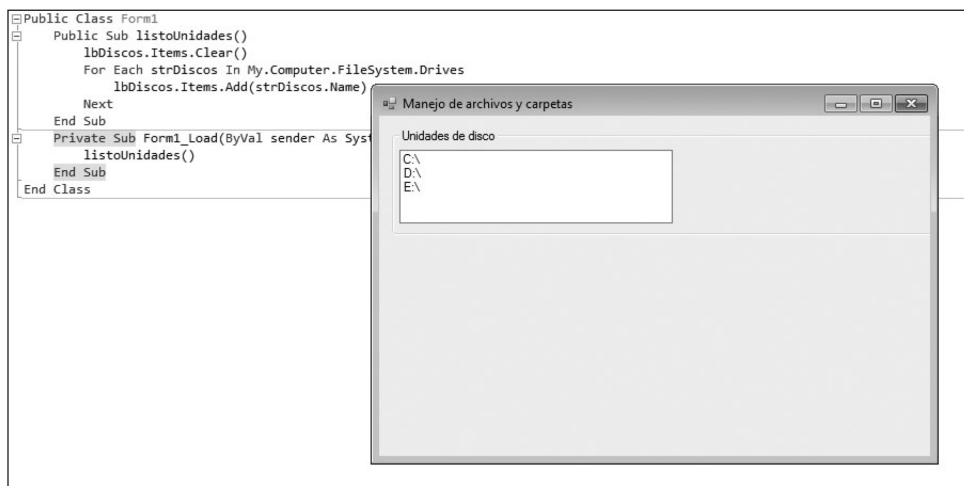


***Figura 22.** Las unidades de disco de nuestra computadora, listadas con tan solo cinco líneas de código.*

Archivos y carpetas

Con el mismo namespace que nos permite listar las unidades de disco, también podemos ver los contenidos de carpetas y archivos. Para ver su funcionamiento, agregamos al formulario de trabajo dos **ListBox** más. uno llamado **lbCarpetas** y otro, lbArchivos. Recordemos siempre explorar con tiempo y detenimiento cada namespace que utilizamos, para aprender los secretos de funciones que no se listan en ningún libro o manual. Disponemos los controls de manera vertical sobre **Form1**, justo debajo de **GroupBox**. A continuación, escribimos el siguiente procedimiento:



PANEL CONTENEDOR GROUPBOX

La idea de utilizar paneles contenedores como **GroupBox** permite no solo agrupar controles en nuestra aplicación, sino también ahorrar varias líneas de código si es que debemos cambiar el estado **Enabled** de dichos controles a False por algún motivo. Con solo modificar la propiedad **GroupBox.Enabled = False**, podremos bloquear el acceso a todos los controles contenidos.

```
    Public Sub listoCarpetas(ByVal strUnidad As String)
        lbCarpetas.Items.Clear()
        For Each strCarpetas In My.Computer.FileSystem.GetDirectories(strU
nidad)
            lbCarpetas.Items.Add(strCarpetas.Remove(0, 3))
        Next
    End Sub
```

Este procedimiento cambia respecto al anterior, dado que debe recibir un parámetro **strUnidad**, desde el cual listar las carpetas. **My.Computer.FileSystem.GetDirectories()** es la función que permite obtener una colección de todas las carpetas de la unidad indicada a través de **strUnidad**. Para agregar las carpetas al control **lbCarpetas**, debemos tener en cuenta que la colección recibida incluirá, al principio del string, la unidad seguida de la carpeta, con lo cual debemos tener la precaución de eliminarla antes de listar el nombre.

La función **Remove()** incluida en el string **strCarpetas** permite especificar qué porción de texto remover de dicho string. Esto se hace indicándole como parámetro la posición inicial del string, desde donde deberá eliminar, seguido de la cantidad de caracteres por quitar.

Por último, llamamos dicho procedimiento desde el evento **lbDiscos _SelectedIndexChanged()**, pasándole como parámetro **lbDiscos.Text**. Luego de este código, ejecutamos nuestro proyecto presionando **F5**:

```
Private Sub lbDiscos_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles lbDiscos.SelectedIndexChanged
        listoCarpetas(lbDiscos.Text)
End Sub
```

Vemos, paso a paso, cómo nuestro nuevo proyecto va tomando vida y funcionalidad con tan sólo muy pocas líneas de código.



USOS DE BARRAS DE HERRAMIENTAS

Si bien las barras de herramientas estilo **Ribbon**, como las incluidas en Office 2007 y 2010, no fueron oficializadas en Visual Studio 2010, a partir de Windows 8, todas las aplicaciones y las ventanas del sistema operativo dispondrán de ellas. Seguramente, en la próxima versión de Visual Studio ya se contemple incluirlas como un control.

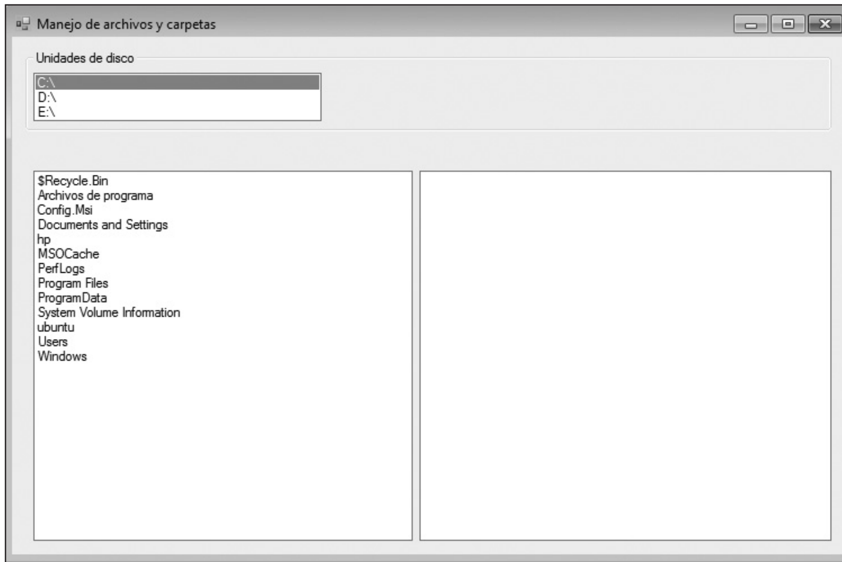


Figura 23. *Al hacer clic sobre una unidad listada, podemos ver cómo se muestran las carpetas que contiene.*

Sólo nos falta hacer clic en una carpeta listada y, sobre el control **listBox** de la derecha, mostrar todos los archivos que contiene. Sigamos, a continuación, agregando a nuestro proyecto el siguiente procedimiento:

```
Public Sub listoArchivos(ByVal strRuta As String)
    lbArchivos.Items.Clear()
    For Each strArchivos In My.Computer.FileSystem.GetFiles(strRuta)
        lbArchivos.Items.Add(strArchivos)
    Next
End Sub
```

Y luego llamemos este procedimiento desde **lbCarpetas_SelectedIndexChanged()**, pasándole como parámetros la unidad y la carpeta seleccionadas:

```
Private Sub lbCarpetas_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles lbCarpetas.SelectedIndexChanged
    listoArchivos(lbDiscos.Text & lbCarpetas.Text)
End Sub
```

Ejecutamos otra vez nuestro proyecto pulsando **F5** para ver cómo quedó. Seguramente, nos encontraremos con la sorpresa de que, en **lbArchivos**, se listan todos los

archivos de la carpeta seleccionada, pero con el mismo problema que tuvimos la precaución de que no nos suceda dentro del procedimiento **listoCarpetas()**: los archivos se listan con la ruta completa. Si vamos a utilizar un simple control ListBox en lugar de un control TreeView o ListView, podemos solucionar este tema aplicando aplicar la función **strArchivos.remove()**, teniendo en cuenta de tomar previamente en una variable el valor de **strRuta.Length + 1**.

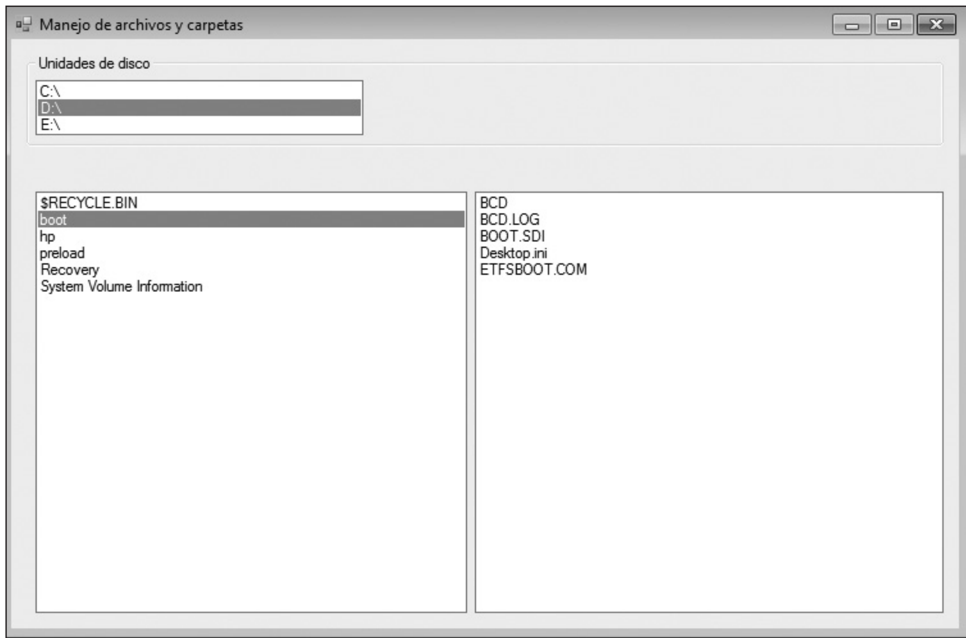


***Figura 24.** Listar archivos y carpetas de nuestra computadora es una tarea sencilla gracias a los **namespaces** de .NET Framework.*

Buscar un archivo

Iniciamos un nuevo proyecto **Application Windows Forms** llamado **BuscarArchivos**. Le agregamos dos controles **Label**, un **TextBox**, un control **Button** y un **ListBox**. A continuación, configuramos sus propiedades según se indica en la **Tabla 7**.

CONTROL	NOMBRE	PROPIEDAD	VALOR
TextBox	txtBuscarArchivos	Text	(Vacío)
Label	lblWildCard	Text	Archivo(s) que contiene(n):
Label	lblArchivoResultado	Text	Resultado de la búsqueda: 0
Button	Button1	Text	Buscar
ListBox	lbFilesResult	Text	(Vacío)

***Tabla 7.** Configuración principal de los controles del nuevo proyecto.*

A continuación, incluiremos el siguiente código dentro de **Button1_Click()**:

```
Imports System.IO

Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim f As Long
    lbFilesResult.Items.Clear()
    If Trim(txtArchivoABuscar.Text) <> “” Then
        Dim strBusco As String = Trim(txtArchivoABuscar.Text)
        Dim strRuta As String =
My.Computer.FileSystem.SpecialDirectories.MyDocuments.ToString
        For Each Archivo As String In
My.Computer.FileSystem.GetFiles(strRuta, FileIO.SearchOption.SearchAllSubDi
rectories, _
                            strBusco)
            Refresh()
            lbFilesResult.Items.Add(Archivo)
        Next
        f = lbFilesResult.Items.Count
        lblArchivoResultado.Text = “Resultado de la búsqueda: “ & f & “
archivo(s) encontrado(s)”
    End If
End Sub
End Class
```

La misma función **GetFiles** que utilizamos en el ejemplo anterior para listar los archivos de la carpeta seleccionada se usa en este caso con una serie de parámetros adicionales, que nos permiten especificar en qué ruta buscar (**strRuta**), si la búsqueda debe realizarse también dentro de los subdirectorios que tenga esta



SPECIAL FOLDERS

My.Computer.FileSystem.SpecialDirectories nos provee de una colección de **carpetas especiales** de Windows. En este **namespace** podemos listar fácilmente las carpetas como **Desktop**, **MyDocuments**, **MyPictures** y **ProgramFiles**, entre otras, de manera automática, sea cual sea el perfil de usuario que utilice dicha computadora en el momento de ejecutar la aplicación.

carpeta (**FileIO.SearchOption.SearchAllSubDirectories)** y la cadena que contiene el tipo de archivo por buscar (**strBusco**).
Los resultados obtenidos serán listados en **lbFilesResult**, y la variable **f** contará cuántos ítem fueron agregados a este control.

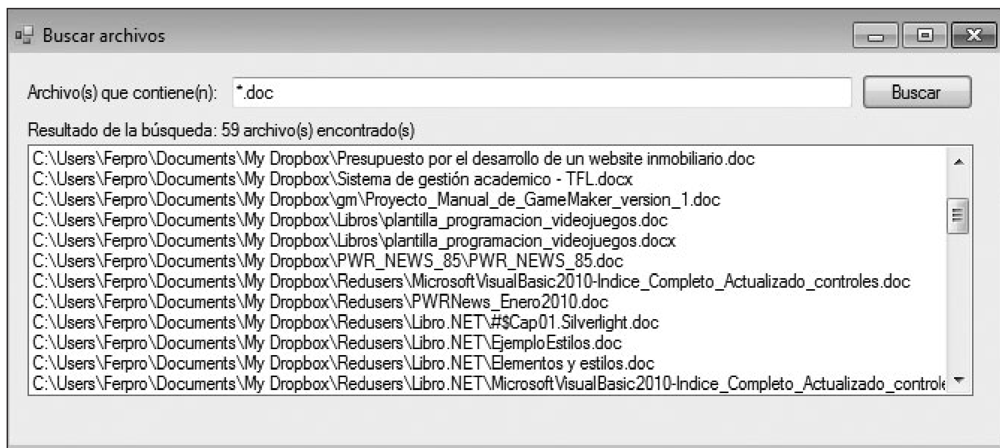


***Figura 25.** Resultado de la búsqueda de documentos de Word en la carpeta **Documentos** de nuestro perfil.*

FolderBrowseDialog

El control **FolderBrowseDialog** permite que, a través de un cuadro de diálogo, seleccionemos un directorio específico entre las carpetas y unidades de nuestro sistema, para luego realizar una operación determinada. Iniciamos un nuevo proyecto **Application Windows Forms**, llamado **FolderBrowseDialog**. Le agregamos un **Botón**, un **TextBox** y el control **FolderBroseDialog**, al que llamaremos **FolderbDG**.
Dentro de **Button1_Click()** escribimos el siguiente código:

```
FolderbDG.RootFolder = Environment.SpecialFolder.MyDocuments
FolderbDG.Description = “Seleccione una carpeta”
```



USO DE WILDCARDS

Al buscar un archivo, podemos utilizar caracteres comodín si queremos traer todos los archivos de un mismo tipo ***.DOC**, o todos los que tengan un nombre específico. Estos comodines se denominan **wildcards** y fueron heredados del antiguo **DOS**, cuando especificábamos un parámetro de búsqueda o la copia de un conjunto de archivos hacia otro destino.

```
FolderbDG.ShowNewFolderButton = False
FolderbDG.ShowDialog()
If Windows.Forms.DialogResult.OK Then
    TextBox1.Text = FolderbDG.SelectedPath
End If
```

RootFolder permite indicar en qué carpeta se posicionará la vista al abrir el cuadro de diálogo, **Description** se usa para establecer un título para la ventana, y **ShowNewFolderButton** permite indicar si aparece el botón **Crear nueva carpeta**. Presionamos **F5** a continuación para ver el resultado del proyecto:

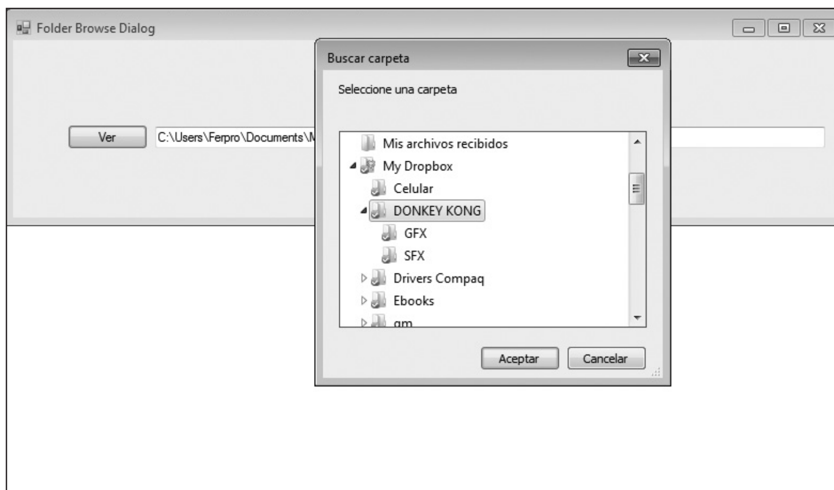


***Figura 26.** La ruta completa de la carpeta seleccionada se mostrará en TextBox1.*

Verificar si existe un archivo o directorio

FileExists() y **DirectoryExists()** son dos funciones que se encuentran en **My.Computer.FileSystem** y que nos permiten verificar si existe un archivo o carpeta



LIMITACIONES DEL CONTROL RTF

El control **RichTextBox** no contempla características básicas, como soporte de impresión directa ni preview para los archivos **.RTF**. Esto se debe a que Microsoft prioriza el uso de WordPad como editor de texto básico. La firma no libera estas características ni el soporte para documentos **.DOC** por una cuestión de estrategia de negocios.

en una determinada ruta. En un nuevo proyecto **Application Windows Forms**, agregamos dos **label** y dos **button**. En un **label** especificamos la ruta de un archivo, y en el otro, la ruta de un directorio. A continuación, añadimos el siguiente código en los eventos **Click()** de **Button1** y **Button2**:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim msj As String
    Dim bExiste As Boolean = My.Computer.FileSystem.FileExists(Label1.Text)
    If bExiste Then
        msj = "El archivo especificado sí existe."
    Else
        msj = "El archivo especificado no existe."
    End If
    MessageBox.Show(msj, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Excla
mation)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim msj As String
    Dim bExiste As Boolean =
My.Computer.FileSystem.DirectoryExists(Label2.Text)
    If bExiste Then
       msj = "El directorio especificado sí existe."
    Else
       msj = "El directorio especificado no existe."
    End If
    MessageBox.Show(msj, Me.Text, MessageBoxButtons.OK,
MessageBoxIcon.Exclamation)
End Sub
```



PROCESAR DOCUMENTOS DE WORD

Si deseamos potenciar una aplicación con soporte para abrir y guardar documentos **.DOC**, podemos optar por controles de terceras partes, como los de **www.independentsoft.de/office/index.html**. Esta empresa brinda APIs que permiten manipular todo tipo de documentos de Word por un costo mucho menor a la licencia del producto en sí.



Figura 27. *Probemos a ingresar archivos o carpetas inexistentes en nuestro sistema Windows, para verificar el correcto funcionamiento de las funciones.*

Crear directorios

Otra función muy útil que encontramos en **My.Computer.FileSystem** es **CreateDirectory()**, a través de la cual podemos crear un directorio. Esta función lleva un parámetro que indica dónde se debe crear el directorio:

```
My.Computer.FileSystem.CreateDirectory("C:\Visual Basic 2010")
```

Copiar archivos

También disponemos en el mismo **namespace** de la función **CopyFile()**, que nos permite copiar archivos. Puede contener una serie de sobrecargas que alteren su acción según un evento determinado:

```
My.Computer.FileSystem.CopyFile(strOrigen, strDestino)
```



MANEJO DE ARCHIVOS Y CARPETAS

Es conveniente que, para llevar a cabo todo el trabajo sobre archivos y carpetas, dispongamos de un directorio específico donde previamente copiemos el contenido de toda nuestra computadora. Luego de hacerlo, podremos efectuar todas las pruebas necesarias sin correr el riesgo de perder información que puede ser vital para nosotros.

strOrigen indica la ruta origen del archivo que queremos copiar, en tanto que **strDestino** indica la de destino. Otras sobrecargas que permite son, por ejemplo, **OverWrite**. Este es un boolean para indicar que, si llega a existir el archivo en el destino, hay que sobrescribirlo o no. **ShowUI**, también booleano, determina si se muestra una ventana de progreso de la copia mientras esta se realiza.

```
My.Computer.FileSystem.CopyDirectory(strOrigen, strDestino)
```

La función **CopyDirectory** hace lo propio con los directorios, y posee las mismas sobrecargas que **CopyFile** que nos aseguran cuidar todos los aspector durante el proceso.

Borrar archivos

DeleteFile() permite eliminar archivos de nuestro disco:

```
My.Computer.FileSystem.DeleteFile(strArchivo)
```

Posee sobrecargas que permiten indicar si se debe mostrar una ventana de progreso mientras el archivo se elimina, y otra para especificar si el archivo debe ser enviado a la Papelera de reciclaje o tiene que ser eliminado directamente.

```
My.Computer.FileSystem.DeleteDirectory(strDirectory, FileIO.DeleteDirector
yOption.DeleteAllContents)
```

DeleteDirectory() permite hacer lo propio con una carpeta de directorio. Esta función posee una sobrecarga donde podemos especificar si, en caso de que haya contenido dentro del directorio por eliminar, este se borre también o no.

Renombrar archivos

RenameFile() permite cambiar el nombre de un archivo por otro. Posee dos parámetros: **ArchivoDeOrigen** y **NuevoNombreDeArchivo**. **RenameDirectory()** permite hacer lo propio con un directorio con tán sólo dos parámetros también.

Propiedades de un archivo

Todos los archivos poseen propiedades que permiten obtener información de ellos. Podemos acceder a ellas a través de **GetFileInfo()**.

```
Dim information = My.Computer.FileSystem.GetFileInfo(“C:\pagefile.sys”)
```

```
MsgBox("El nombre completo es " & information.FullName & ".")
MsgBox("Último acceso: " & Information.LastAccessTime & ".")
MsgBox("Tamaño: " & Information.Length & ".")
```

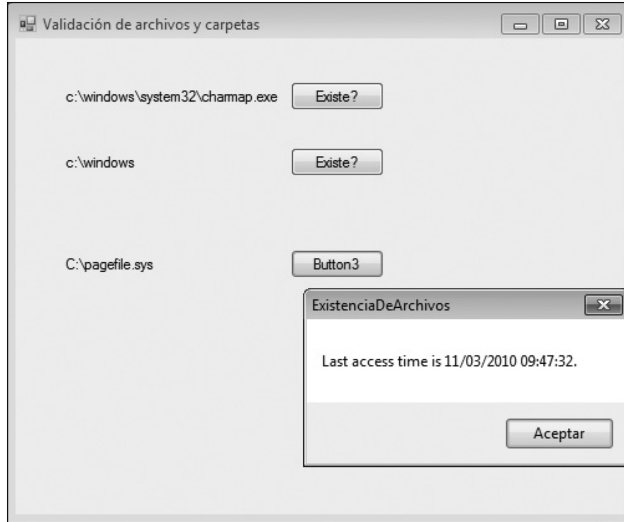


Figura 28. *El resultado de **GetFileInfo** es devuelto en un arreglo, y mediante él, obtenemos todas las propiedades del archivo consultado.*



RESUMEN

Hemos realizado un repaso general por el uso de archivos y carpetas de nuestro sistema a través de Visual Basic 2010. También vimos cómo escribir, leer y grabar archivos de texto plano y de texto enriquecido. Es conveniente aprender bien el manejo de archivos, dado que el uso de todas estas funciones casi siempre será aplicado sobre algún desarrollo importante. Las implementaciones son infinitas: un administrador de archivos y carpetas, un sistema de backup, un procesador de textos, y tantas otras posibilidades más a las que el namespace My.Computer.FileSystem nos abre las puertas. Podemos seguir explorando todas las otras funciones que ofrece el espacio de nombres para así acrecentar más nuestro conocimiento sobre él.



ACTIVIDADES

PREGUNTAS TEÓRICAS

1. Investigue algunos ejemplos para implementar los controles **TreeView** y **ListView** leyendo el contenido de archivos y carpetas.

2. Arme una ventana de propiedades de archivo y agregue los controles correspondientes para crear una ventana de propiedades similar a la que ofrece el sistema operativo.

3. Dentro del Editor de texto enriquecido, complete la función **Buscar** sobre el texto, reemplazando la implementada en el Editor mediante un **For**, por la función **IndexOf()**.

4. También complete la función faltante de **Cambiar color de fondo** del Editor de texto enriquecido, reutilizando el control **FontDialog()** que agregamos en el proyecto.

5. Agregue un **MDIForm** al proyecto de Editor de texto enriquecido, para poder crear múltiples documentos de texto.

6. Modifique la función **Buscar** del punto 3, para convertirla en una función de búsqueda y reemplazo.

7. Teniendo presente el concepto de la construcción de menús, incluya en el Editor de texto enriquecido el control **ContextMenuStrip** para activar el botón derecho del mouse sobre el control RichTextbox.

8. Estudie el resto de las propiedades y funciones que se incluyen en el control **RichTextbox** para potenciar más el Editor de texto enriquecido.

9. Investigue la clase **Process.Start()**.

10. Investigue el evento **RichTextbox_LinkClicked()** e incluya la clase **Process.Start()** para abrir los hipervínculos que se presentan en los documentos **RTF**.

Bases de datos

En este capítulo haremos una introducción a uno de los principales factores que se aplican en todo desarrollo de sistemas, la conexión, administración y manejo de información en bases de datos. Para esto debemos aprender, como primera medida, qué es una base de datos, las tablas y registros que la componen; luego veremos las distintas opciones de bases de datos que existen, qué herramientas nos brinda el IDE para interactuar con ellas y los registros, y de qué modo conectarse desde Visual Basic 2010 para manejar la información.

INTRODUCCIÓN

Desde mediados de 1980, los sistemas informáticos de gestión comenzaron a popularizarse. Las empresas decidieron aprovechar el poder de cálculo de las computadoras, y las incorporaron de a poco en el trabajo diario de sus empleados. Para implementar su uso y lograr que este fuera productivo, debieron estructurar la información por almacenar en diversos paquetes de software junto con programadores. De esta manera, los sistemas de gestión comenzaron a tener su auge y, por consiguiente, las bases de datos también entraron en escena. Las empresas ya contaban (y hoy siguen contando) con lo que se conoce como almacenes de datos. Los depósitos donde se alojan las facturas, órdenes de compra, órdenes de fabricación y los libros contables, entre otra papelería de uso diario, se denominan bancos de datos. Estos han existido desde siempre y seguirán existiendo por mucho tiempo más, aunque en estos últimos años, numerosas compañías se han volcado a digitalizar sus documentos para lograr, por un lado, disponer de ellos de manera perpetua, algo que el papel no puede garantizar; y por otro, tener la posibilidad de ahorrar muchos metros cúbicos de espacio.

Qué es una base de datos

Se conoce como **base de datos** a un conjunto de datos de un mismo tipo que son almacenados de manera sistematizada para disponer de ellos en cualquier momento. Podemos considerar como ejemplo claro de lo que es una base de datos a una biblioteca, que se compone en su mayoría de documentos en papel categorizados de modo que cualquier persona pueda consultarlos con facilidad. En la realidad moderna, y gracias al desarrollo y avance de la tecnología informática, las bases de datos actuales se ofrecen en un formato electrónico digital, y pueden prestar soluciones amplias para cualquier tipo de problema. Hace algunas décadas, cuando estas comenzaron a popularizarse, solo podían prestar servicio de almacenamiento de datos escritos. Hoy en día, permiten guardar desde registros de datos, hasta imágenes, videos, documentos y software.
Para el manejo de una base de datos se crearon los **sistemas de gestión**, conocidos por su abreviatura en inglés, **DBMS** (*DataBase Management System*). Estos



STORED PROCEDURES

Los Stored Procedures, o procedimientos almacenados, permiten realizar operaciones en SQL Server de la misma manera que utilizamos código SQL embebido en la aplicación, con la ventaja de que estas se procesan en el lado del servidor, y la aplicación no requiere estar conectada a la fuente de datos a la espera de un resultado.

permiten realizar las operaciones fundamentales sobre una base de datos, desde su creación, diseño de estructura de tablas, campos y registros, hasta la consulta, inserción, actualización y eliminación de datos.

```
CURSOR   <-- -->     UP   DOWN        DELETE          Insert Mode:  Ins
 Char:    ‹   ›    Record:  ↑    ↓     Char:    Del    Exit:         ^End
 Field: Home End   Page:   PgUp  PgDn  Field:   ^Y     Abort:        Esc
 Pan:     ^←  ^→   Help:    F1         Record:  ^U     Set Options: ^Home

SNO PART_NO---  PART_NAME---------------------  QTY SRC MTL REMARKS--
 1  1901.0009   STUD FOR FLY WHEEL MAGNETO FLANGE  7
 2  2201.0063   OIL LEVER PLUG                     3
4   4103.0084   try                               24
 4  4204.2001
 5  4204.2002   DIFFERENTIAL HOUSING COVER         1
 6  4204.2003   GASKET
 7  4204.2005   BEARING                            1
 8  4204.2006   BUSH FOR DIFFERENTIAL PINION ROLLER
 9  4204.2007   NIDDLE CAGE
10  4204.2011   SHIMS FOR REAR AXLE                1
11  4204.2016   WASHER FOR PIN JOINT              16

BROWSE          ||<C:>||REAR AXL              ||Rec: 1/65      ||      ||
                        View and edit fields.
```

***Figura 1. Dbase** es una de las primeras bases de datos en operar en sistemas de computadoras bajo el entorno **DOS**.*

Toda aplicación desarrollada para trabajar hoy maneja de alguna manera una base de datos interna, donde almacena la información o la configuración que el usuario aplica para, luego, poder disponer de ella de la manera más cómoda posible. En el ámbito cotidiano de cualquier empresa u organismo público, también se utilizan a diario las bases de datos. En una compañía se guarda la información contable o de producción de los artículos o servicios que esta brinde. En los organismos públicos, suelen utilizarse, por ejemplo, para contabilizar los datos demográficos de la población, como nombres de ciudadanos, domicilio y número de documento. También las empresas de servicios organizan su trabajo en grandes bases de datos. Las ciudades ya desarrolladas disponen de sistemas de bases de datos para la dirección del tránsito y manejo de semáforos; las compañías de servicios de agua y luz también conservan la información de sus clientes en ellas, como nombres, domicilio y contabilización del consumo de los medidores, y gracias a esto, pueden luego calcular la facturación que deben hacer mensual o bimestralmente. Existen distintos **tipos de bases de datos**. En la **Tabla 1** se presenta cada uno de ellos, con la información que puede albergar.

TIPO DE BASE DE DATOS	GESTIÓN BRINDADA
Bases de datos estáticas	Son bases de solo lectura, usadas para guardar información histórica. Son consultadas para estudiar el comportamiento de un conjunto de datos a través del tiempo y, luego, poder tomar decisiones.
Bases de datos dinámicas	Son bases donde la información almacenada sufre modificaciones con el tiempo. Permiten realizar operaciones de consulta, borrado o modificación de registros.

TIPO DE BASE DE DATOS	GESTIÓN BRINDADA
Bases de datos bibliográficas	Contienen información de la fuente primaria para poder localizarla. Un ejemplo es una base de datos bibliográfica, con autores, fechas de publicaciones, editoriales y ediciones de determinadas publicaciones, junto con un probable resumen del contenido de cada una de ellas.
Bases de datos de texto completo	Almacenan una fuente primaria, como el contenido de todas las ediciones de una colección de revistas científicas o de saber popular.
Bases de datos de directorio	Almacenan, por ejemplo, un directorio de teléfonos de una ciudad.

***Tabla 1.** Algunos de los distintos tipos de bases de datos que podemos encontrar y sus respectivos usos.*



***Figura 2.** La **Biblioteca Nacional Argentina** dispone de un ejemplo de base de datos estática, con imágenes del país agrupadas por categorías.*

Por otro lado, las bases de datos también se dividen en modelos, que indican la administración de información. Los modelos son descripciones de la abstracción



HISTORIA DE LAS BASES DE DATOS

El ingeniero Herman Hollerit creó la primera base de datos dentro de una máquina perforadora de tarjetas, utilizada en 1890 para procesar los datos obtenidos del segundo censo en los EE.UU. Demoró dos años y medio en entregar los resultados, de los cuales se pudo obtener la población infantil y el número de familias existentes hasta entonces.

de su contenido de datos, como también el tipo de método utilizado para almacenar y recuperar los registros. En la **Tabla 2** encontramos los distintos modelos de bases de datos y sus usos más comunes.

MODELO DE BASE DE DATOS	APLICACIONES MÁS COMUNES
Jerárquicas	Almacenan la información en una estructura jerárquica o de árbol, donde un nodo denominado padre puede tener varios nodos hijos. Son útiles para aplicaciones que deben manejar un gran volumen de información y datos compartidos.
De red	Difieren muy poco del modelo jerárquico, dado que cambia su concepto de nodo: un mismo nodo puede tener varios nodos padres. Este modelo mejoró la posibilidad de redundancia de datos, que el jerárquico no podía manejar de manera óptima.
Transaccionales	Permiten enviar y recibir datos a altas velocidades. No son muy comunes, y se utilizan más en el ámbito del análisis de la calidad y para datos de producción industrial.
Relacionales	Este modelo es el más utilizado en la actualidad para administrar los datos de forma dinámica. Se creó en 1970 en los laboratorios de IBM y se consolidó con un nuevo paradigma de los modelos de base de datos ideal para relacionar la información contenida. Fue el primer modelo en utilizar lo que se conoce como normalización de datos. En la década del 80, Dbase fue la primera base de datos en utilizar este modelo.
Multidimensionales	Este modelo fue ideado para desarrollar aplicaciones que permiten, por ejemplo, la creación de cubos OLAP. No son muy distintas de las bases de datos relacionales.
Orientadas a objetos	Este modelo es muy reciente, y se aplica mucho en los tiempos que corren. Almacenan los objetos completos, con su estado y comportamiento. Utilizan encapsulación, herencia y polimorfismo.

***Tabla 2.** Si bien existen otros modelos de bases de datos, los aquí listados son los más usados en las últimas décadas.*

Estructura de almacenamiento de datos

Las bases de datos deben estar estructuradas de tal manera que toda la información que contengan sea de fácil acceso. Para esto, su estructura debe dividirse correctamente en **tablas**, **campos** y **registros**.

Tablas

En una base de datos, una **tabla** es un medio donde se concentra la información de los campos y registros de manera lógica. Cada base puede contar con una o miles de tablas, dependiendo de la complejidad de la información que almacene.

Campos

Los **campos** se almacenan dentro de las tablas, y contienen la información sobre el tipo de datos que se guardará en ellos: numéricos, alfanuméricos o binarios; a su

vez, de cada uno de ellos se pueden desprender nuevos tipos. Vemos en la **Tabla 3** qué tipos de datos pueden contener los campos.

TIPO DE DATOS	USO COMÚN
Numérico	Pueden ser enteros o reales. Los primeros son números sin decimales, y los segundos pueden contener decimales.
Booleanos	Solo poseen dos estados: verdadero o falso.
Memo	Es un campo alfanumérico de longitud ilimitada.
Fechas	Permiten almacenar fechas para ordenar de manera cronológica o calcular períodos de tiempo.
Alfanuméricos	Admiten números y letras. Se limitan a 255 caracteres como máximo.
Autoincrementables	Comúnmente son campos numéricos que incrementan su unidad de valor por cada nuevo registro agregado. Sirven como identificador único de cada registro en una tabla.

***Tabla 3.** En resumen, estos son los tipos de datos que puede almacenar una tabla. De cada uno de ellos pueden desprenderse más tipos, dependiendo de la precisión buscada.*

Existen convenciones determinadas para los tipos de datos que se almacenarán en cada campo. Si bien un campo de tipo alfanumérico puede contener solo números, estos no podrían ser nunca procesados por el lenguaje de la base de datos para entregar, por ejemplo, una suma total, porque al no ser un tipo de campo numérico, no podrá ser sumado.

Registros

Se denomina registro al conjunto de información que se almacena en una tabla, indicando cada dato en su campo correspondiente. En una tabla denominada **Personas**, es posible almacenar los datos de varias personas (Nombre, Apellido, Documento, Domicilio, Teléfono, E-mail). Cada una que tenga sus datos allí almacenados será considerada como un registro.

Empleados

Id	Compañía	Apellidos	Nombre	Dirección de correo electrónico	Cargo	Teléfono de	Teléfono pa	Teléfono m	Número
1	Northwind Traders	González	María	nancy@northwindtraders.com	Representante de venta	987 654 321	987 654 321		987 654 3
2	Northwind Traders	Escolar	Jesús	andrew@northwindtraders.com	Vicepresidente de vent	987 654 321	987 654 321		987 654 3
3	Northwind Traders	Pinilla Gallego	Pilar	jan@northwindtraders.com	Representante de venta	987 654 321	987 654 321		987 654 3
4	Northwind Traders	Jesús Cuesta	María	mariya@northwindtraders.com	Representante de venta	987 654 321	987 654 321		987 654 3
5	Northwind Traders	San Juan	Patricia	steven@northwindtraders.com	Jefe de ventas	987 654 321	987 654 321		987 654 3
6	Northwind Traders	Rivas	Juan Carlos	michael@northwindtraders.com	Representante de venta	987 654 321	987 654 321		987 654 3
7	Northwind Traders	Acevedo	Humberto	robert@northwindtraders.com	Representante de venta	987 654 321	987 654 321		987 654 3
8	Northwind Traders	Bonifaz	Luis	laura@northwindtraders.com	Coordinador de ventas	987 654 321	987 654 321		987 654 3
9	Northwind Traders	Chaves	Francisco	anne@northwindtraders.com	Representante de venta	987 654 321	987 654 321		987 654 3
(Nuevo)									

***Figura 3.** En el ejemplo vemos una tabla denominada **Empleados**, con sus respectivos campos y registros almacenados.*

Qué son los índices

Los índices son campos que permiten asociar una tabla con una o varias columnas de otras tablas. Determinan una relación entre el contenido y el número de fila

donde está ubicado el registro. Estos tipos de campos permiten agilizar las consultas a las tablas, y evitar así que el motor de la base de datos deba revisar uno a uno los registros hasta dar con el que estamos buscando para devolver un conjunto de resultados. Cuando creamos una tabla, lo ideal es que en ese momento establezcamos el campo índice. Un índice puede contener una o más columnas por indexar. En el ejemplo de la **Figura 3** podemos determinar que el índice puede darse a través del campo **Id**, el cual establece un número único para cada empleado cargado. También podríamos denominar como índice al campo **Apellido**, ya que en este caso, todos los apellidos son distintos. No sería un buen ejemplo denominar al campo **Nombre** como índice, porque, en este caso, encontramos a dos personas con el mismo nombre. Para los motores de bases de datos, un índice ágil generalmente es el denominado por un dato numérico. **Id** en este caso sería el índice óptimo, aunque también podríamos denominar como índice a la combinación de dos campos, **Apellido** y **Nombre**. Es más factible en este ejemplo que este último índice funcione, dado que no se repiten nombres, pero, por ejemplo, en una base de ciudadanos de un país, los apellidos y los nombres allí almacenados se repetirán muchas veces. En ese caso, el índice correcto para esa tabla sería el campo **Nro. De documento**.

Qué son las vistas

Las bases de datos suelen estar compuestas por muchas tablas. Por lo general, debemos combinar información de distintas tablas para mostrar en pantalla o, simplemente, necesitamos mostrar algunos campos de una tabla. Para llevar a cabo estas acciones, siempre es conveniente establecer una **Vista de datos** que nos permita acceder de manera rápida a estos registros. En los motores de bases de datos más populares esto se conoce como **Vistas**. Las vistas agrupan los campos que necesitamos ver de una o más tablas en un conjunto de datos de solo lectura. Dentro de este conjunto de datos, también podemos especificar ciertos filtros si es que no precisamos mostrar todos los registros. Por ejemplo, si tenemos una base de datos de empleados de la que solo queremos obtener el nombre y apellido de aquellos que pertenecen a un determinado sector, debemos armar una vista que permita filtrar dicha información.



LENGUAJE DE CONSULTA ESTRUCTURADO

SQL es el modelo de lenguaje de consulta para las bases de datos desde hace varias décadas. Distintas bases de datos, como **SQL Server**, **MySQL**, **PostgreSQL** y **Access**, utilizan en su motor el lenguaje de consultas SQL, aunque cada una realizó modificaciones sobre él. De todos modos, la esencia básica de consulta es similar para todas.

Empleados

Id	Compañía	Apellidos	Nombre	Dirección de correo electrónico	Cargo	Teléfono de	Teléfono pa	Teléfono m
1	Northwind Traders	González	María	nancy@northwindtraders.com	Representante de ventas	987 654 321	987 654 321	
3	Northwind Traders	Pinilla Gallego	Pilar	jan@northwindtraders.com	Representante de ventas	987 654 321	987 654 321	
4	Northwind Traders	Jesús Cuesta	María	mariya@northwindtraders.com	Representante de ventas	987 654 321	987 654 321	
6	Northwind Traders	Rivas	Juan Carlos	michael@northwindtraders.com	Representante de ventas	987 654 321	987 654 321	
7	Northwind Traders	Acevedo	Humberto	robert@northwindtraders.com	Representante de ventas	987 654 321	987 654 321	
9	Northwind Traders	Chaves	Francisco	anne@northwindtraders.com	Representante de ventas	987 654 321	987 654 321	
(Nuevo)								

***Figura 4.** Una vista de la tabla **Empleados**, donde solamente se muestran los recursos con el cargo **Representante de Ventas**.*

Qué es una entidad relación

Las bases de datos conllevan un modelo de **Entidad-Relación**, a través del cual se logra normalizar la información contenida para que sea lo menos redundante posible. Esto permite establecer una dinámica en la respuesta sobre las consultas de datos. Este modelo se sustenta en diferentes conceptos que permiten representar como resultado un modelo de la vida real.

Entidad

La entidad representa un objeto del mundo real independiente que se diferencia de otros similares aunque sea del mismo tipo de sus pares. Un claro ejemplo de esto sería una persona. Cada persona se diferencia de las demás por sus rasgos o características que la hacen única. Hasta en el caso de gemelos, cada uno se distingue del otro por el nombre o número de documento. Estas diferencias de entidades se conocen como atributos, que se explican en el apartado siguente.

Atributos

Los atributos, entonces, son propiedades que describen a una entidad en un conjunto de entidades. Cada entidad dispone de valores específicos asignados a sí misma para lograr una identificación unívoca. Un conjunto de personas se diferencia de sus pares por un conjunto de atributos específicos, como nombre, apellido, número de documento, estatura, peso, edad, etcétera.

Relación

Una relación equivale a una cierta dependencia entre las entidades. Un conjunto de personas, cada una con atributos distintos, pueden estar relacionadas entre sí, por ejemplo, por su nacionalidad o porque trabajan en una misma compañía.

Relaciones entre tablas

Todas las tablas de una misma base de datos pueden relacionarse entre sí a través de uno o más campos que permitan representar una relación de la misma naturaleza. Las reglas de una relación pueden mantener restricciones, que no deben quebrantarse a menos que haya una relación de una tabla de un solo registro con muchos registros de otra tabla. El conjunto de relaciones del que participan dos o más entidades

se establece con la correspondencia de cardinalidad, la cual indica el número de entidades con la que puede estar relacionada una sola entidad. Suponiendo que tenemos un conjunto de entidades denominadas A y B, entre ellas puede haber más de un tipo de cardinalidad. Veamos en detalle la cardinalidad en la **Tabla 4**.

TIPO DE RELACIÓN	DESCRIPCIÓN
Uno a uno	Una entidad de A se relaciona de manera directa con una entidad de B (y viceversa).
Uno a varios	Una entidad de A se relaciona con ninguna o muchas entidades de B, pero una entidad de B se relaciona con una única entidad de A.
Varios a uno	Una entidad de A se relaciona de manera exclusiva con una entidad de B, y una entidad de B puede relacionarse con ninguna o muchas entidades de A.
Varios a varios	Una entidad de A se relaciona con ninguna o muchas entidades de B (y viceversa).

Tabla 4. *Un detalle de los tipos de cardinalidad existente en una base de datos.*

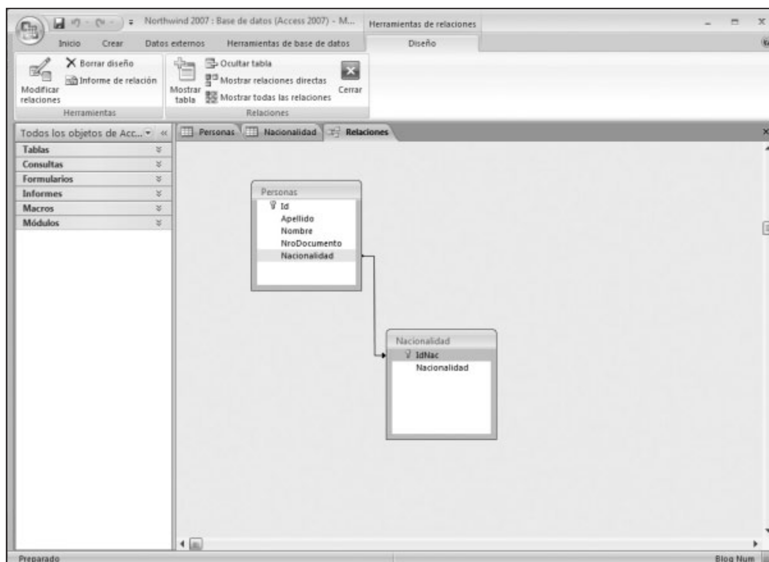


Figura 5. *En el ejemplo vemos una relación entre una tabla **Personas** y una tabla **Nacionalidad**.*



UML

El lenguaje **UML** (*Unified Modelling Language*) permite modelar los datos de sistemas de software. Este lenguaje gráfico da la posibilidad de visualizar, especificar y construir un sistema a través de estándares, e incluir aspectos conceptuales de los procesos de negocios, funciones, expresiones de lenguajes de programación y DB Schemas para obtener componentes reutilizables.

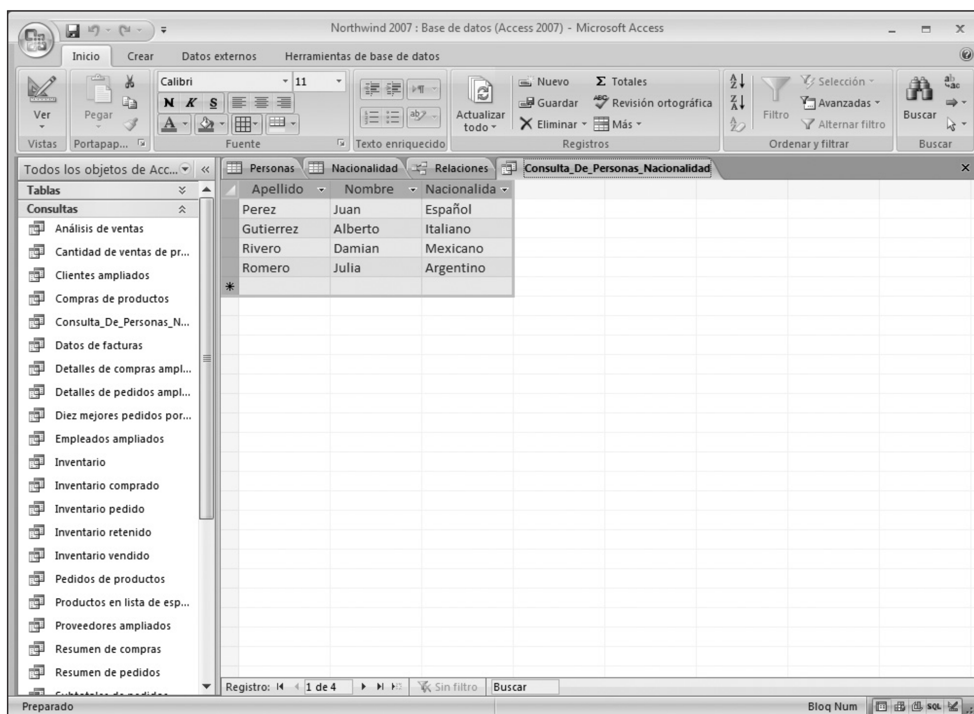


Figura 6. *En la vista obtenemos una relación de* ***uno a uno****, aunque si se agregara un nuevo registro con una nacionalidad ya asignada, esta cambiaría a* ***varios a uno****.*

BASES DE DATOS Y VISUAL BASIC

Visual Basic 2010, como sus versiones anteriores, está preparado para trabajar con bases de datos. El entorno de desarrollo está listo para conectarse y realizar operaciones con dos bases de datos populares propias de **Microsoft: SQL Server** y **Access**. SQL Server es un motor de base de datos profesional, que debe adquirirse por separado, aunque desde su versión 2005, Microsoft decidió lanzar al mercado una versión Express gratuita del motor de base de datos, orientada más



LIBROS EN PANTALLA

SQL Server posee una ayuda vía web, conocida como Libros en pantalla. Es conveniente configurar de manera predeterminada la ayuda a través de la Web si es que disponemos de una conexión permanente. Podemos acceder a la ayuda desde el siguiente link: **http://msdn.microsoft.com/es-ar/library/ms130214.aspx**.

a la práctica de los usuarios. Claro que con ella también podemos llevar adelante desarrollos con ciertas limitaciones, como el hecho de que la base no puede tener un tamaño mayor que **4 GB**, y el tipo de conexión de datos debe ser de manera directa, no pudiendo acceder a ella en forma remota. Para los ejemplos que se llevarán adelante en este capítulo, **SQL Server 2008 Express Edition** cumplirá con creces nuestras necesidades de aprendizaje. Para instalarla, ingresamos en el link de la página de **Microsoft** y realizamos la descarga del instalador: **www.microsoft.com/downloads/details.aspx?displaylang=es&FamilyID=58ce885d-508b-45c8-9fd3-118edd8e6fff**. El tamaño es de aproximadamente 90 MB, con lo cual tomará un tiempo realizar el proceso. Una vez concluido este paso, ejecutamos el **Setup** de **SQL Server 2008**.

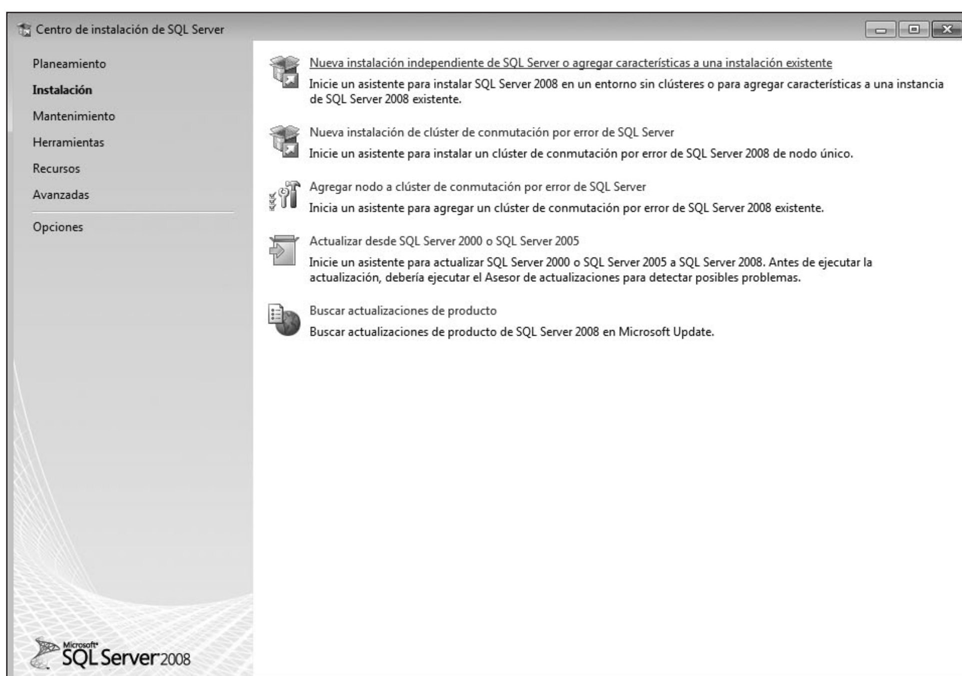


Figura 7. *La ventana de instalación de la base de datos* ***SQL Server 2008****. A través de ella, podemos verificar si nuestra computadora cumple con todos los requisitos.*

INSTALACIÓN DE SQL SERVER

Ya sea que utilicemos **SQL Server 2005** o **2008**, siempre será conveniente descargar el instalador del motor de base de datos y el administrador gráfico correspondiente, y copiar los instaladores a un **medio óptico** o **pendrive**. Si en el futuro debemos reinstalar la computadora, no tendremos que perder tiempo volviendo a buscar y descargar ambos instaladores.

El programa de instalación revisará nuestra computadora para verificar que cumple con los requisitos necesarios para ejecutar SQL Server 2008 sin problemas. Si faltara algún componente, el programa de instalación sugerirá descargarlo e instalarlo de manera automática. Luego de esto, nos pedirá seleccionar las características de SQL Server que deseamos instalar; elegimos solamente **Servicios de motor de base de datos**. Al finalizar la instalación, procedemos a descargar **SQL Server Management Studio Express**, la interfaz gráfica a través de la cual crearemos las bases de datos, las tablas y las vistas. Realizamos la descarga desde el link: **www.microsoft.com/downloads/details.aspx?familyid=08E52AC2-1D62-45F6-9A4A-4B76A8564A2B&displaylang=es.** Luego, ejecutamos el programa de instalación, y seguimos los pasos que se nos van indicando.

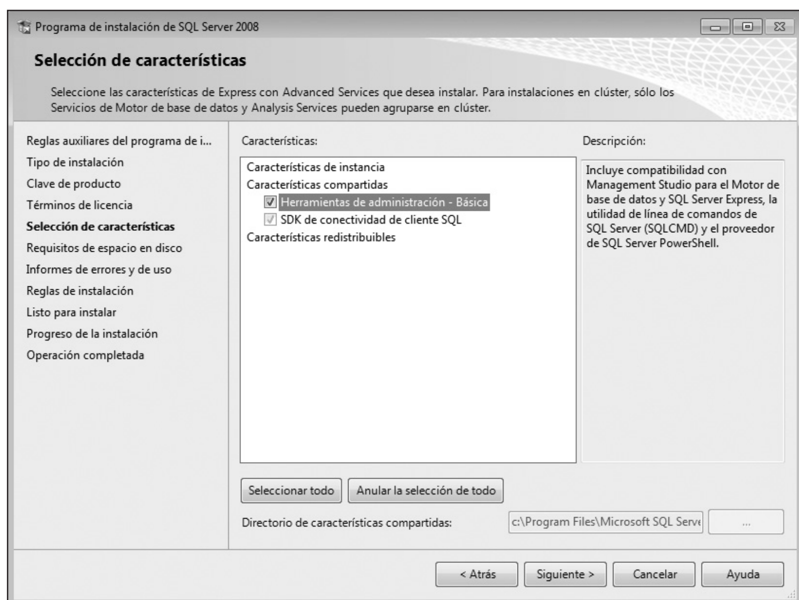


***Figura 8.** El proceso de instalación de **SQL Server Management Studio** es similar al de SQL Server 2008.*

Una vez instalados SQL Server 2008 y Management Studio, iniciamos sesión en él para diseñar una base de datos, algunas tablas y ver un poco el manejo de datos a través del lenguaje SQL. Para hacerlo, buscamos en la carpeta de programas **SQL Server 2008** el icono correspondiente a Management Studio e ingresemos en él. Encontraremos un panel izquierdo llamado **Explorador de objetos**, desde donde podremos acceder a todos aquellos que componen el motor de base de datos. Sobre el ítem **Bases de datos** hacemos un clic con el botón derecho del mouse y, del menú contextual, seleccionamos **Nueva base de datos**. Se abrirá una ventana en donde podremos especificar el nombre de la base, su tamaño inicial (**en megabytes**) y la ruta donde se guardará el archivo físico. Podemos dejar los parámetros que ofrece

SQL Server por defecto. En **Nombre de la base de datos** ingresamos **ejemplo**, y en **Ruta de acceso** especificamos **c:\databases**. Es conveniente almacenar las bases de datos en una carpeta que tengamos siempre a mano; si no lo hacemos, SQL Server las guardará en el **%directorio de instalación%\MSSQL10.SQLEXPRESS\MSSQL\DATA**.

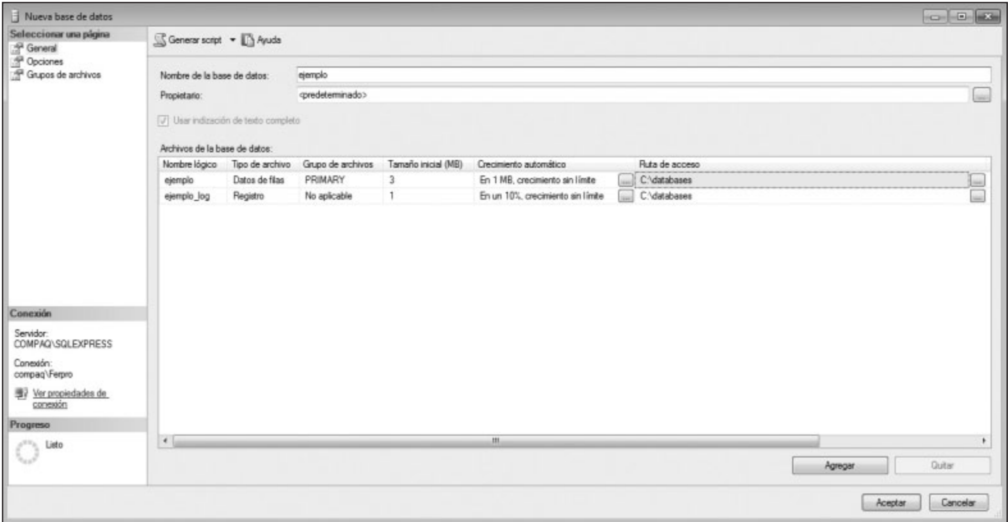


***Figura 9.** Estos son los parámetros iniciales para la creación de una nueva base de datos.*

En la pestaña **Opciones** podemos configurar parámetros adicionales, como el modelo de recuperación de datos en caso de tener que restaurar un backup, el nivel de compatibilidad con bases de datos más antiguas (**SQL Server 2005** y **2000**) y la Intercalación, que nos permite adaptar la base de datos con todas sus tablas para aceptar caracteres de otros idiomas, además de los occidentales.

Crear tablas en SQL Server

Una vez que generamos la base de datos, procedemos a crear dos tablas. Para hacerlo, expandimos las opciones anidadas sobre la base de datos, hacemos clic derecho en la carpeta **Tablas** y, del menú contextual, seleccionamos **Nueva Tabla**. Agregamos los campos según se especifica en la **Tabla 5**.

NOMBRE DE COLUMNA	TIPO DE DATOS	PROPIEDADES DE COLUMNA
IDPersona	bigint	Especificación de Identidad: Sí Permitir valores nulos: No
Apellido	nvarchar(50)	Permitir valores nulos: No
Nombre	nvarchar(50)	Permitir valores nulos: No
Empresa	nvarchar(50)	Permitir valores nulos: No

***Tabla 5.** Creación de la tabla Contactos.*

Al finalizar la creación de la tabla, sobre el campo **IDPersona** hacemos clic derecho y, del menú contextual que emerge, elegimos **Establecer clave principal**. Luego guardamos la tabla con el nombre **contactos**. Repetimos el procedimiento para crear una nueva tabla, ingresando los datos especificados en la **Tabla 6**.

NOMBRE DE COLUMNA	TIPO DE DATOS	PROPIEDADES DE COLUMNA
IDReunion	Bigint	Especificación de Identidad: Sí Permitir valores nulos: No
Fecha	Date	Valor o enlace predeterminado: GetDate() Permitir valores nulos: No
Hora	Time	Permitir valores nulos: No
Contacto	Bigint	Permitir valores nulos: No
Motivo	Text	Permitir valores nulos: No

***Tabla 6.** Creación de la tabla Citas.*

Al finalizar la creación de la tabla, repetimos el paso anterior, pero esta vez haciendo clic con el botón derecho sobre el campo **IDReunion**, para seleccionar luego **Establecer clave principal** (ver figura a continuación).

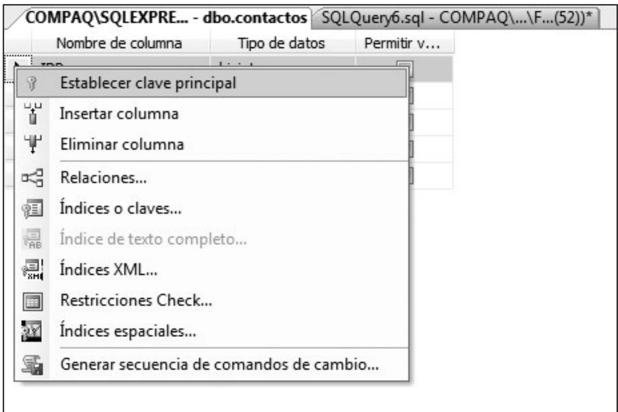


***Figura 10.** El menú contextual siempre se ajusta al evento sobre el cual estamos trabajando; en este caso, la creación de la tabla contactos.*



COMENTARIOS SOBRE CÓDIGO SQL

Al igual que en los lenguajes de programación, SQL Server nos permite realizar comentarios antes o después de cada sintaxis escrita, iniciando la línea con dos guiones seguidos, y luego, especificando el comentario. Esto es válido tanto para los comentarios realizados en la ventana de instrucciones como para los que hagamos dentro de un procedimiento almacenado.

Guardamos la tabla con el nombre **citas**. Con estas dos tablas nos bastará para comprender un poco la sintaxis del lenguaje de consulta estructurado con la cual podremos trabajar sobre los datos almacenados en la base.

SENTENCIA INSERT

Probemos a continuación cómo agregar datos a una de las tablas que creamos. Para hacerlo, en la barra de herramientas de SQL Server Management Studio, presionamos en el primer botón, denominado **Nueva Consulta**. Se abrirá una ventana donde podremos escribir la sentencia para agregar un registro a una tabla existente. Copiamos el código que se muestra a continuación:

```
USE ejemplo
INSERT INTO contactos (Apellido, Nombre, Empresa)
VALUES (Luna, ‘Fernando’, ‘F-DIGITAL’)
```

El comando **USE** nos permite asegurarnos sobre qué base de datos vamos a trabajar. Luego, viene la sentencia que agregará un registro a la tabla **contactos**; su sintaxis es casi una oración en inglés: insertar en la tabla **contactos**, dentro de los campos **Apellido**, **Nombre**, **Empresa**, los valores **Luna**, **Fernando**, **F-DIGITAL**. Como estamos insertando un tipo de datos texto, debemos especificarlo entre comillas. Presionando **CTRL + F5**, podemos analizar si la sintaxis escrita es correcta o no.
Luego, al pulsar **F5** se ejecutará la instrucción. Como mensaje devuelto en la ventana de resultados, debemos leer la leyenda **Comandos completados exitosamente**. A continuación, repetimos la sentencia **INSERT** cambiando los valores que queremos guardar por lo siguiente, expresado en la línea de código:

```
USE ejemplo
INSERT INTO contactos (Apellido, Nombre, Empresa)
VALUES (‘Moreno’, ‘Mariano’, ‘Otra empresa S.A.’)
INSERT INTO contactos (Apellido, Nombre, Empresa)
VALUES (‘Conte’, ‘Gabriel’, ‘Italia s.r.l.’)
```

SENTENCIA SELECT

Para poder ver los tres registros agregados a la tabla **contactos** hasta el momento, debemos utilizar el siguiente comando:

```
USE ejemplo
SELECT * FROM contactos
```

Ejecutamos esta instrucción presionando **F5**: veremos los tres registros insertados. Para ver específicamente determinados campos, en vez de poner **asterisco**, escribimos los nombres de los campos separando por una coma:

```
SELECT Apellido, Nombre FROM contactos
```

Si queremos redistribuir el orden de cada campo, sólo debemos realizar el procedimiento anterior indicando el orden que queremos darles a los campos:

```
SELECT Empresa, Nombre, Apellido FROM contactos
```

Y si queremos ordenar los registros de determinada manera, solo debemos especificar el o los campos por los cuales queremos ordenar:

```
—- orden ascendente

SELECT * FROM contactos ORDER BY IDPersona
— Presionar F5 para ver resultados

—- orden descendente

SELECT * FROM contactos ORDER BY IDPersona DESC
— Presionar F5 para ver resultados
```

SENTENCIA UPDATE

Para modificar algún dato de un registro ya ingresado, utilizamos la sentencia **UPDATE**. Modifiquemos a continuación los registros 2 y 3:

```
UPDATE contactos SET Nombre = ‘Nicolas’, Empresa = ‘ALBERTAX S.A.’ WHERE
IDPersona = 2

UPDATE contactos SET Nombre = ‘Julian’, Empresa = ‘Cittadella Construc
ciones’ WHERE IDPersona = 3
```

Esta sentencia modifica los campos **Nombre** y **Empresa** del registro cuyo **IDPersona** sea igual a **2** e igual a **3**. Si no especificamos la cláusula **WHERE**, los cambios se verán reflejados en todos los registros de la tabla **contactos**.

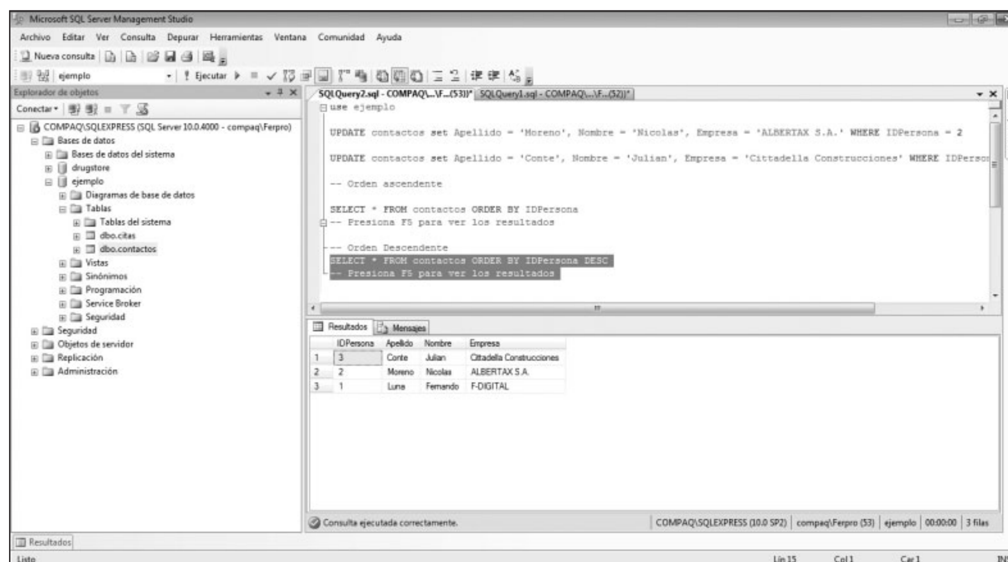


***Figura 11**. Así vemos las sentencias escritas hasta el momento. Para no eliminar las anteriores, podemos seleccionar la sentencia por ejecutar antes de presionar **F5**.*

SENTENCIA DELETE

Agregaremos a continuación tres registros más, para luego probar la sentencia **DELETE**. Copiamos el código a SQL Server Management Studio y lo ejecutamos:

```
INSERT INTO contactos (apellido, nombre, empresa) VALUES ('BENVENUTTO',
'PEPE', 'Famiglia s.r.l.')
INSERT INTO contactos (apellido, nombre, empresa) VALUES ('PEREZ GARCIA',
'JUAN CARLOS', 'Serie TV s.a.')
INSERT INTO contactos (apellido, nombre, empresa) VALUES ('SAMBUSETTI',
'Jorge', 'Ponea s.r.l.')
```

Ahora tenemos un total de seis registros en la tabla **contactos**. A continuación, repasaremos la sentencia **DELETE**, cuya estructura para eliminar un registro es la siguiente:

```
DELETE tabla WHERE campo = valor
```

Tal como cuando utilizamos la sentencia **SELECT** con uno o más parámetros, de la misma manera debemos especificar el o los parámetros necesarios que nos permitirán eliminar uno o más registros de la tabla.

Eliminemos a continuación un registro cuyo campo **apellido = 'Perez Garcia'**. Para hacerlo, escribimos la siguiente sentencia:

```
DELETE contactos WHERE Apellido = 'PEREZ GARCIA'
```

Seleccionamos la sentencia y presionamos **F5**. Luego, volvemos a ejecutar la sentencia **SELECT** para actualizar los datos de la tabla. Veremos que dicho registro fue eliminado, y ahora solo nos quedan cinco. Ejecutamos otra vez la instrucción **INSERT**, agregando este registro eliminado nuevamente. A continuación, copiamos la instrucción y la agregamos, modificando solamente el campo **empresa**, cuyo valor es **Canal de aire s.r.l.** Tenemos dos registros casi idénticos en la tabla **contactos**; solo cambia el valor de un campo, que es lo que los diferencia, sacando el **IDPersona**. Probemos, entonces, a ejecutar la última sentencia **DELETE** otra vez, luego eliminará ambos campos. Para los casos en que necesitamos eliminar uno determinado y en una tabla hay muchos iguales que se repiten, no debemos especificar como parámetro de eliminación los campos comunes que pueden contener registros repetidos, sino que debemos utilizar un campo índice, cuyo valor sea único e irrepetible en toda la tabla. Vamos a agregar dos registros con los mismos valores en todos sus campos:

```
INSERT INTO contactos (apellido, nombre, empresa) VALUES ('PEREZ GARCIA',
'JUAN CARLOS', 'Serie TV s.a.')
INSERT INTO contactos (apellido, nombre, empresa) VALUES ('PEREZ GARCIA',
'JUAN CARLOS', 'Serie TV s.a.')
```

Ejecutamos la sentencia **SELECT** y vemos que los últimos dos registros que se agregaron son exactamente iguales, excepto por su campo **IDPersona**. Para eliminar el último registro repetido, en este caso utilizamos la siguiente sentencia:

```
DELETE contactos WHERE IDPersona = 8
```



SENTENCIAS CON MÚLTIPLES PARÁMETROS

Podemos usar cualquier sentencia con la cláusula **WHERE** especificando uno o más parámetros. Por ejemplo: **campo = valor AND campo2 = valor**, etc. También es posible utilizar operadores como **NOT** e **IN**, y combinar varios con múltiples campos. Además, son válidas las cláusulas **=**, **<**, **>**, **>=**, **<=** y **LIKE**; esta última, combinándola con un **carácter comodín**.

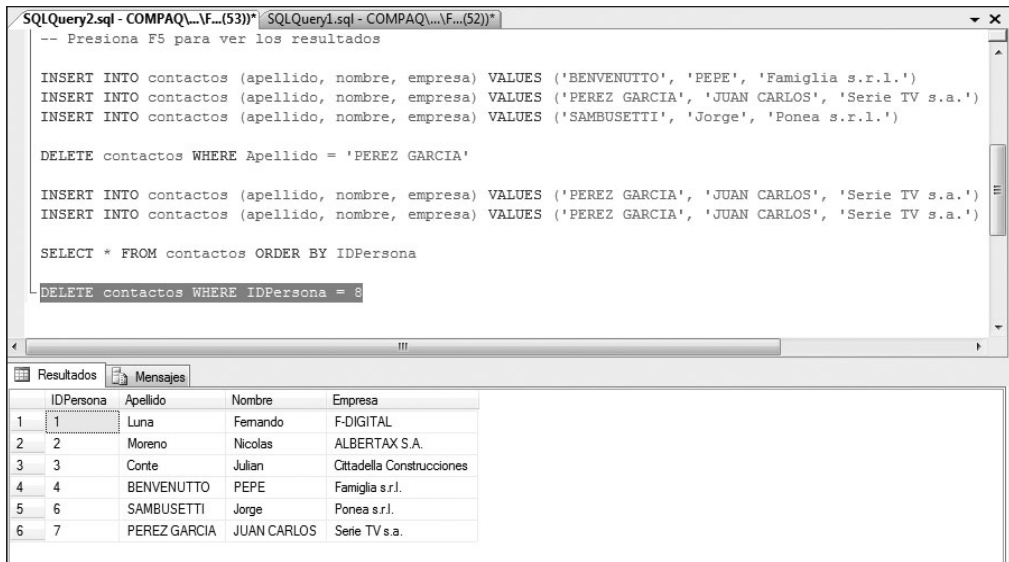


***Figura 12.** Con esta última sentencia, vemos que el resultado de la sentencia **DELETE** solo se encargó de eliminar un único registro duplicado.*

Hasta ahora hemos visto cómo manejarnos con **SQL Server 2008** y **SQL Server Management Studio** en su propio entorno. Ya que conocemos la manera de crear una base de datos y sus tablas, y sabemos operar con las sentencias más comunes para el manejo de registros, iniciaremos la introducción al manejo de bases de datos desde el IDE de Visual Basic 2010. Tanto este lenguaje como el resto de los integrantes de Visual Studio .NET pueden realizar conexiones con las bases de datos más populares del mercado. Visual Basic es capaz de interactuar sin problemas con SQL Server y Access de manera nativa, y a través de drivers conectores, con cualquier otro tipo de motores.

El Explorador de base de datos

Para manipular las conexiones con bases de datos de manera cómoda, y poder ver las **tablas**, **consultas** y **procedimientos almacenados**, entre otras funciones disponibles, Visual Basic cuenta con un Explorador de bases de datos, que permite administrar desde el mismo IDE todo el contenido de la o las bases con las cuales estemos trabajando en un proyecto. A su vez, contamos con la pestaña **Orígenes de datos**, que muestra el o los **dataSets** a través de los cuales establecemos conexión con el motor de base de datos para operar sobre su contenido. Para centralizar el manejo de manera cómoda desde Visual Basic, la misma ventana de **Propiedades** que nos muestra la información relacionada con los controles permite interactuar también con los objetos de la base de datos, y nos muestra también las propiedades de cada objeto que la compone. De esta manera,

podemos consultar cualquier campo y el tipo de datos que establecimos al diseñar correctamente nuestra base de datos.

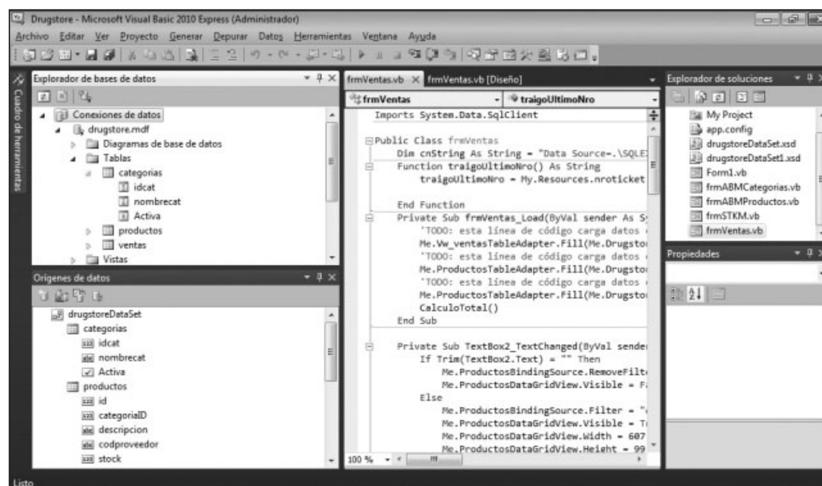


Figura 13. *En la figura podemos apreciar las tres ventanas que nos permiten visualizar todo el contenido de una base de datos, sin tener que recurrir a SQL Server Management Studio.*

CONECTAR Y TRABAJAR CON BASES DE DATOS DESDE VB.NET

La tecnología utilizada por Visual Basic y por el resto de los lenguajes que componen la Suite Visual Studio se denomina **ADO.NET**. El nombre **ADO** proviene de la abreviatura de **ActiveX Data Objects**, la cual nace a partir de la versión **5.0** de Visual Basic. Esta tecnología engloba un conjunto de componentes que permiten a los programadores realizar conexión con bases de datos y, por consiguiente, operar con ellas, al dar la opción de realizar lo que se conoce como



DIVERSIDAD DE BASES DE DATOS

Visual Basic mantiene la facilidad de conectarnos con cualquier origen de datos, ya sea a través de los **Data Providers**, que habitualmente son propietarios de cada compañía que comercializa las bases, o mediante **ODBC**. Microsoft provee un **SDK** para desarrollar los drivers necesarios y poder interactuar con cualquier motor desde .NET.

CRUD (**CReate**, **Update**, **Delete**). ADO.NET conforma la biblioteca de clases base incluida en el Framework .NET. La arquitectura de ADO.NET se basa en dos componentes principales: **Data Provider** y **DataSets**.
Data Provider: es una clase que brinda lo necesario para establecer una conexión a una fuente de datos, como **SQL Server**, **Access** y **Oracle**. Cada una de estas fuentes contiene su propio conjunto de objetos individual, pero todas incluyen clases de utilidades en común. Dentro de esta clase encontramos algunos objetos que se mustran en la **Tabla 7**:

OBJETO	FUNCIÓN PRINCIPAL
Connection	Permite establecer una conexión a una fuente de datos para comunicarse desde nuestra aplicación.
Command	Permite realizar lectura de datos, actualizaciones o eliminación de registros.
Parameter	Permite describir un parámetro para utilizar en el objeto Command. Se utiliza para trabajar con procedimientos almacenados y realizar consultas o modificaciones en los registros de una o más tablas.
DataAdapter	Funciona como interconexión para la transferencia de datos entre una fuente de datos y un objeto DataSet.
DataReader	Se utiliza para manipular de manera óptima una lista de resultados obtenidos a través de una consulta.

Tabla 7. *Las **clases** y **objetos** principales para manipular el acceso a bases de datos desde .NET.*

DataSets: este objeto permite manipular las clases que interactúan con la información de una base de datos en memoria. El objeto **DataSet** representa un esquema de datos entero o parte de él, e incluye las tablas y las relaciones entre sí. Contiene un objeto **DataTable**, que representa una tabla perteneciente a la base con la cual se estableció la conexión. Su estructura está conformada por un **nombre de tabla**, las **filas** y **columnas**, cada una con sus respectivas **propiedades**. El objeto **DataView** se desprende de un **DataTable**, ordenando los datos a través de la cláusula **ORDER BY** de **SQL**. También es el encargado de administrar los filtros, desde la cláusula **WHERE** del lenguaje de consulta. Los objetos **DataTables** tienen un filtro por defecto, y para lograr otros filtros adicionales, es preciso recurrir a la creación de tantos **DataViews** como sean necesarios. Esto permitirá reducir la consulta reiterada con la base de datos y, así, mejorar la performance de la aplicación. También están los objetos componentes de **DataViews** y **DataTables**, que despliegan en profundidad todo el contenido de la tabla o consulta que estamos accediendo. **DataColumn** representa una columna de la tabla, con su nombre y tipo de datos. **DataRow** representa una sola fila del conjunto de registros de la tabla. **DataRowView** representa una sola fila de un **DataView**. **DataRelation** es la representación de la relación entre dos o más tablas a través de sus claves. **Constrait** es una propiedad de la base de datos que se debe cumplir de manera obligatoria, como así también así también los valores únicos para una columna que oficia de clave primaria.

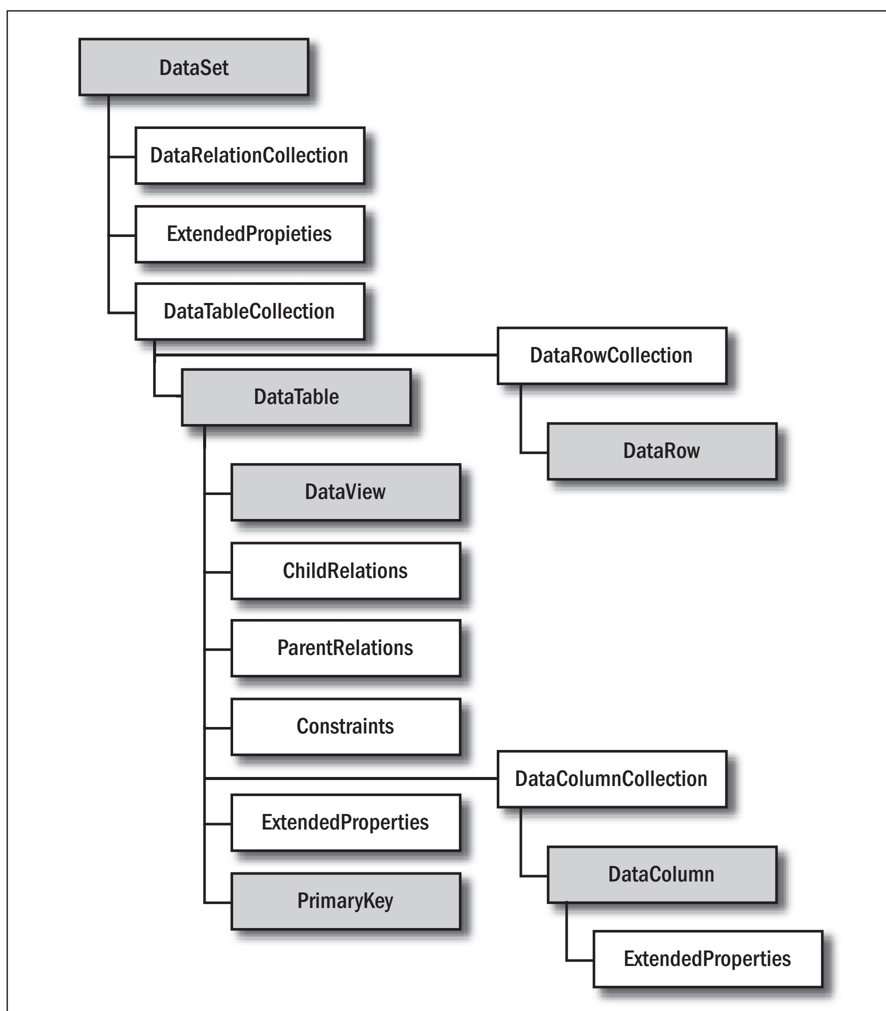


***Figura 14.** Parte de la estructura que compone el acceso a datos del framework .NET.*

Controles para manejar una base de datos

Visual Basic ofrece dos tipos de conexión a bases de datos: a través de los objetos y controles, utilizando asistentes de conexión, lo cual acorta mucho el proceso de diseño de una aplicación; y a través de los objetos, mediante código. Esta última forma requiere mucho más tiempo de desarrollo, pero permite controlar al máximo todos los detalles necesarios en la manipulación de datos.

La pestaña Datos del Cuadro de herramientas

En el **Cuadro de herramientas** de Visual Basic 2010 encontramos un **Grupo** denominado **Datos**, que contiene objetos y controles destinados a establecer conexión

con la base de datos que también nos permiten operar con sus registros, así, mostrar el o los registros de datos en pantalla.

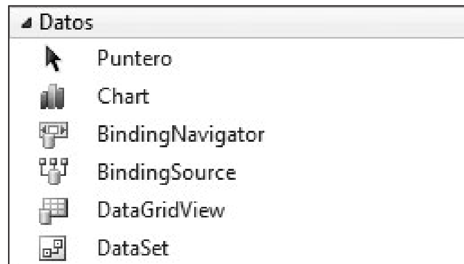


Figura 15. *El grupo Datos del **Cuadro de herramientas** agrupa los controles y objetos necesarios para la conexión con fuentes de datos.*

DataSet

El **DataSet** es el principal componente de cualquier aplicación con conexión a datos desarrollada en Visual Basic 2010. Oficia de objeto contenedor de datos en memoria, tal como lo hace cualquier aplicación a través de manejo de caché.

DataGridView

El control **DataGridView** actúa como grilla de datos donde se muestra el resultado de una consulta o conexión a una tabla. Es totalmente personalizable por **columna** y **fila**, lo cual permite configurar al máximo cada **celda**, dependiendo del tipo de datos que contenga. Por ejemplo, para los tipos de **datos numéricos** o de **formato monetario**, es posible establecer la cantidad de decimales, el tipo de símbolo monetario y el separador de miles. El control **DataGridView** puede utilizarse tanto vinculado a una tabla o consulta de datos, como solo. Al utilizarlo de esta última manera, se deberá especificar por código cada fila y columna que se agregue a dicha grilla. Vamos a iniciar un nuevo proyecto del tipo **Application Windows Form** al cual denominamos **EjemploDeDatos**. Sobre el Form1 de nuestro proyecto agregamos un **DataGridView**. Al momento de hacerlo, veremos que se



ORIGEN DE ADO.NET

ADO.NET es visto por algunos de los seguidores de las versiones anteriores a .NET de Visual Studio como un renacimiento de la tecnología **ActiveX Data Objects**. Esto se debe a la completa reestructuración de sus objetos y clases. Otros piensan que es un producto nuevo en vez de la renovación de una arquitectura ya existente.

despliega un menú de acciones denominado **Tareas de DataGridView**, que se aplicará sobre dicho control. Dentro de este menú nos encontramos con la posibilidad de enlazar la grilla a una base de datos, para que nos muestre una **Tabla** o **Vista**. Para este primer ejemplo, no enlazaremos la grilla a ninguna base de datos. En este menú tenemos otras opciones, que nos permiten realizar algunas configuraciones sobre **DataGridView1**: agregar columnas, editar las existentes, habilitar la opción de **Agregar**, **Editar** las filas y **Eliminar**. También contamos con la posibilidad de **Ordenar** los datos desde cada columna, y de **Ajustar** la grilla al ancho y alto del **Form**. Dejamos las opciones tal como están. Sobre la grilla añadimos dos controles **Button**: el primero, etiquetado como **Agregar fila**; y el segundo, como **Eliminar última fila**. Como paso siguiente, del menú **Tareas** de **DataGridView1** seleccionamos la acción **Editar columnas**. Luego, presionamos reiteradas veces el botón **Agregar**. Veremos cómo se van agregando varias columnas a la grilla. Si seleccionamos alguna de ellas, sobre el lateral derecho de esta ventana aparecerá la posibilidad de personalizar dicha columna al máximo: especificar el nombre mostrado en el título de la columna, el ancho, si es solo lectura, si el tipo de datos es personalizado o no, si debe ser de ancho fijo o variable, y si se pueden ordenar, entre otras propiedades más. Sobre el primer botón, en su evento **Click()** agregamos el siguiente código:

```
DataGridView1.Rows.Add()
```

Y sobre el segundo botón, en el mismo evento, escribimos el siguiente código:

```
Dim i As Integer = DataGridView1.Rows.Count - 2
DataGridView1.Rows.RemoveAt(i)
```

En primer lugar, agregamos el código dentro del evento **click()**, que nos permite generar una nueva fila sobre la grilla. El segundo botón permite eliminar específicamente la última fila de la grilla.



CONTROL DATAGRIDVIEW

El control **DataGridView** viene de la combinación de dos controles utilizados en Visual Basic: **DataGrid** y **HierarchicalFlexGrid**. El primero tenía todos los atributos necesarios para conectarse a una tabla o vista de datos y trabajar con los registros fácilmente. El segundo tenía mayor flexibilidad para manejar filas y columnas de manera independiente.

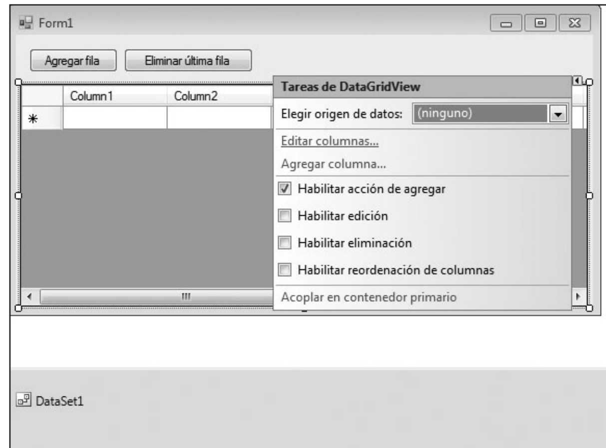


Figura 16. *En esta figura podemos ver el menú de tareas con el que contamos por cada **DataGridView** en nuestro proyecto.*

Debemos tener en cuenta una particularidad de este control: la última fila, si es que no especificamos lo contrario, es un registro nuevo pendiente de confirmación, por lo que no podemos eliminarlo. En caso de que queramos hacerlo, se provocará una **Exception** que nos indicará que la fila en cuestión aún no ha sido confirmada como tal, y por eso no puede ser eliminada. En el código de **Button2** vemos que a la variable **i** le estamos asignando un valor equivalente a la última fila. Para obtener este valor, realizamos un conteo de todas las filas existentes y, sobre este resultado, restamos **2**. La primera posición no la estamos teniendo en cuenta, ya que es una posición de fila sin confirmarla. Ejecutamos nuestro proyecto y probamos a agregar y eliminar filas en el **DataGridView**.

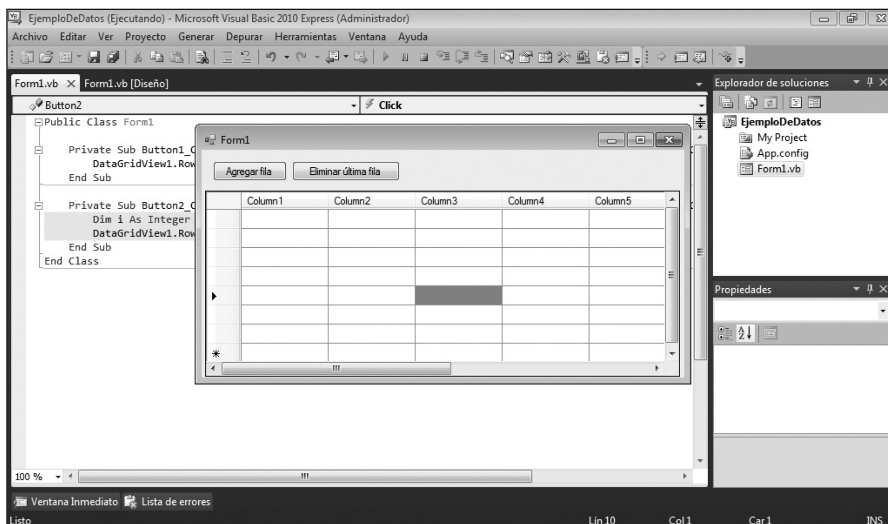


Figura 17. ***DataGridView*** *en acción. Probemos a agregar y eliminar registros con los botones respectivos.*

BindingSource

Este control permite encapsular la información de un conjunto de datos para poder trabajarla luego sobre un formulario. Dentro de la información encapsulada es posible establecer un orden para mostrar los datos y distintos parámetros de filtro. Agregando el control **BindingSource** a nuestro proyecto, encontraremos entre sus propiedades la opción de configurar un **DataSource** u origen de datos. Si seleccionamos nuevo origen de datos del proyecto, mediante un asistente de conexión, podremos configurarlo. Automáticamente, se listarán las bases de datos que ya tenemos configuradas, y podremos establecer otras nuevas que no figuren allí. Seleccionamos **Nuevo conjunto de datos**, y buscamos la base de datos **Ejemplo.mdf**, creada al principio de este capítulo. Una vez hecho esto, probamos la conexión desde esa misma pantalla. Si la conexión a la fuente de datos es correcta, en la pantalla siguiente especificamos el nombre del origen de datos o dejamos el ofrecido por el Asistente. Presionamos **Siguiente** y seleccionamos las tablas, vistas y otros objetos del acceso a datos que podamos utilizar. Al finalizar la selección, podemos agregar el nombre del **DataSet** que se creará, o dejar el sugerido por el Asistente. Finalizamos el asistente para incorporar el **BindingSource**, y ya podremos observar desde la pestaña **Orígenes de datos** en el IDE de Visual Basic 2010. Desde aquí es posible incorporar las tablas y vistas a nuestro proyecto de manera automática. Veamos cómo crear las pantallas de alta, baja y modificación de registros.

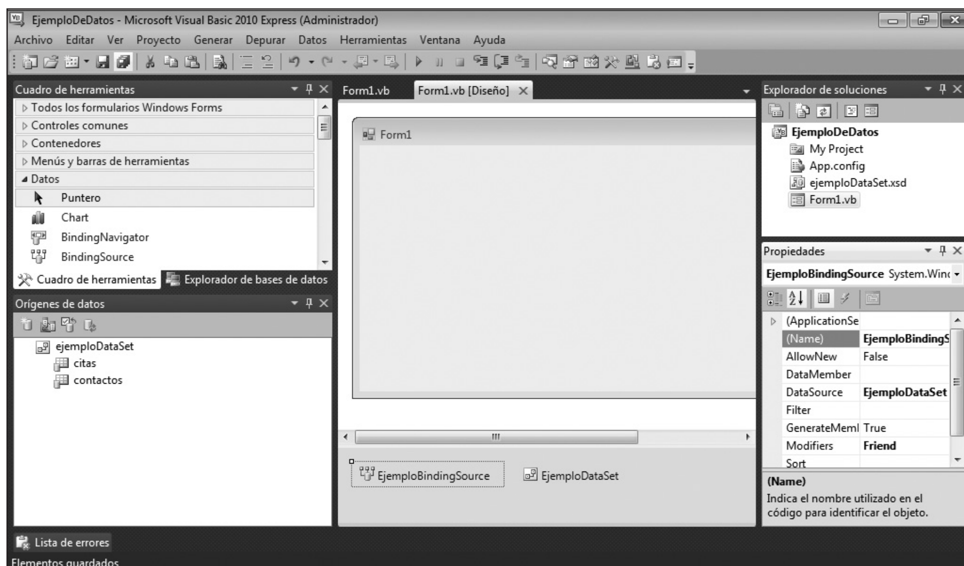


***Figura 18.** En la pestaña **Orígenes de datos** podemos observar las tablas y vistas pertenecientes a la base de datos **Ejemplo.mdf**.*

En la pestaña **Orígenes de datos** veremos dos tablas pertenecientes a la base elegida para conectarnos. Hay una tabla llamada **Citas** y otra denominada **Contactos**.

Borremos de **Form1** lo practicado anteriormente, tanto controles como código agregado, y le agregamos un botón llamado **BtnDetalleCitas** y otro denominado **btnContactos**. Añadimos al proyecto un nuevo Windows Form, al que denominamos **frmContactos**, y como título pondremos **ABM Contactos**. A continuación, añadimos los componentes necesarios para dar de alta nuevos contactos. Para hacerlo, desde la pestaña **Orígenes de datos**, seleccionemos con un clic la tabla **Contactos** y, luego, desplegamos el combo de opciones que nos ofrece. Allí encontraremos la posibilidad de agregar dicha tabla al **frmContactos** de manera automática. Seleccionemos del menú desplegable la opción **Detalles**. Luego, arrastramos la tabla **Contactos** hasta el formulario **frmContactos**. Visual Basic 2010 creará de manera automática todos los componentes necesarios para realizar operaciones con los registros de dicha tabla, tal como lo muestra la **Figura 19**.

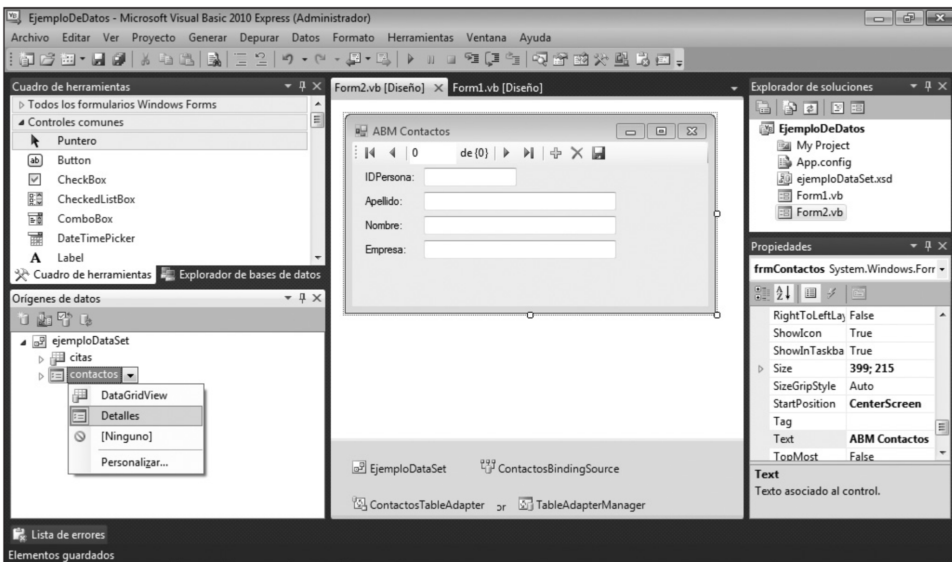


***Figura 19.** Creación automática del **Form ABM** de contactos.*

Nuevamente en **Form1**, en el evento **Click()** del botón **btnContactos** agregamos el código listado a continuación:

```
frmContactos.Show()
```

En **frmContactos**, seleccionamos el primer **textBox** denominado **IDPersonaTextBox** y establecemos su propiedad **Locked = True**. De esta manera, nos aseguramos de que nadie escriba un ID en un campo que genera los índices de manera automática. Por último, nos queda ejecutar la aplicación presionando **F5**, y cargar nuevos contactos y modificar los existentes para probar su funcionamiento.

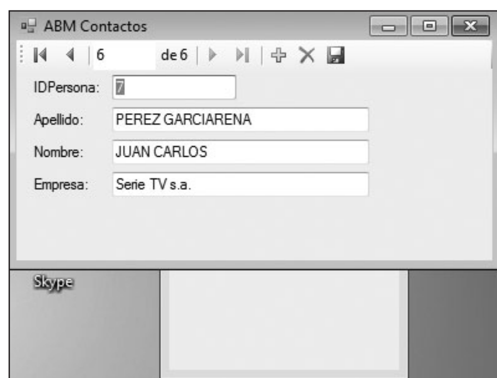


***Figura 20.** frmContactos en acción. Casi sin agregar código, pudimos generar una pantalla para Alta, baja y modificación de datos.*

Sobre el borde superior encontramos una barra de herramientas con los botones necesarios que nos permiten desplazarnos por todos los registros cargados previamente en la tabla **Contactos** y mostrar, a su vez, la cantidad de registros existentes, la que cual generalmente es listado el parcial (registro actual), por sobre el total de la tabla. También veremos botones que nos ayudan a agregar nuevos registros, eliminar otros existentes y guardar las operaciones realizadas. Esta barra de herramientas se conoce como **BindingNavigator**, y la veremos a continuación.

BindingNavigator

Se conoce de esta manera a la interfaz de usuario que permite explorar y operar diferentes controles enlazados a datos. El control **BindingNavigator** permite desplazarse por datos de una tabla o vista, y dar de alta nuevos registros, modificar información de registros existentes, eliminar registros, y guardar los cambios realizados. Este control tiene una serie de propiedades que podemos consultar y configurar para que nos permita, de una manera fácil y rápida, manipular registros en cualquier proyecto que requiera acceso a datos, ya sea a través de la conexión estándar como de la conexión a través de clases creadas específicamente. Veamos sus propiedades y funciones en la **Tabla 8**.



MICROSOFT ACCESS EN VEZ DE SQL SERVER

Microsoft Access permite, casi de manera transparente, utilizar su motor de datos como si se tratase de una base de datos del porte de SQL Server, con la diferencia de que en las sentencias SQL Server es preciso cambiar en determinados casos el carácter comodín, ya sea cuando consultamos un campo fecha o uno de texto.

PROPIEDAD	FUNCIÓN
Name	Permite establecer el nombre del control.
AddNewItem	Desencadena la acción Agregar nuevo registro sobre la tabla a la cual está vinculado.
BindingSource	Establece sobre qué objeto BindingSource se navegará.
CountItem	Permite contabilizar el número total de registros de la tabla o vista.
Delete Item	Permite eliminar el registro donde el objeto BindingSource se encuentra actualmente.
MoveFirstItem	Permite desplazarse hacia el primer registro del objeto BindingSource.
MoveLastItem	Permite desplazarse hacia el último registro del objeto BindingSource.
MoveNextItem	Permite desplazarse hacia el registro siguiente de la posición donde se encuentra actualmente el objeto BindingSource.
MovePreviousItem	Permite desplazarse hacia el registro anterior de la posición donde se encuentra actualmente el objeto BindingSource.
PositionItem	Muestra el número de posición actual donde se encuentra detenido el objeto BindingSource.

Tabla 8. *Propiedades más utilizadas del control BindingNavigator.*

Si, por ejemplo, hacemos doble clic sobre el botón **Guardar** del objeto **Contactos-BindingNavigator**, veremos que en su código se genera de manera automática la acción correspondiente para guardar el registro agregado. **Me.ContactosBindingSource.EndEdit()** aplica sobre la tabla los cambios pendientes al origen de datos. Estos cambios son realizados solamente sobre la caché almacenada en memoria. Por último, **Me.TableAdapterManager.UpdateAll(Me.EjemploDataSet()** refleja los cambios sobre el conjunto de datos de la base física.

CREAR FORMULARIOS CON CONEXIÓN A DATOS

Ya hemos visto de qué manera crear un formulario enlazado a una tabla de datos. Ahora veremos cómo trabajar con un **DataGridView** facilitando el alta, baja y modificación de registros. Como bien hablamos al momento de estudiar el control **DataGridView**, este nos permite trabajar con vinculación a un conjunto de datos o de manera independiente, aunque esta última puede parecer un poco pesada



ACCESO A DATOS EN .NET

Para quienes estén acostumbrados a manejar acceso a datos en **Visual Basic 6**, al principio puede parecer un poco incómoda y confusa esta nueva propuesta generada en .NET para operar sobre bases de datos. La única manera de aprenderla a fondo es practicando varias veces la creación de **Forms** con acceso a datos, tanto enlazados como no enlazados.

si es que queremos crear una conexión a una tabla o consulta de manera manual. Agregamos un nuevo **Windows Form** a nuestro proyecto, llamado **frmCitas**. Dentro de él creamos la pantalla de citas, a través de la cual daremos de alta las nuevas citas relacionadas con un contacto en particular. Desde la pestaña **Orígenes de datos** arrastramos la tabla **Citas**. Se crearán automáticamente una **DataGridView** y un **BindingNavigator**, desde los cuales podremos crear nuevos registros y navegar los existentes.

Operaciones con registros

Una vez creado el **DataGridView**, seleccionamos de él el menú **Tareas**, y luego la opción **Editar Columnas...**. Se abrirá una ventana donde podemos modificar las propiedades de las columnas por defecto. Elegimos la columna **Contacto**. Veremos sobre la derecha todas las propiedades de ella. Modificamos luego las propiedades detalladas en la **Tabla 9**.

PROPIEDAD	VALOR
ColumnType	DataGridViewComboBoxColumn
DisplayStyle	ComboBox
DataPropertyName	Contacto
DataSource	ContactosBindingSource
DisplayMember	Apellido
ValueMember	IDPersona

***Tabla 9.** Cambiando algunas propiedades de las columnas, podemos establecer la celda como un ComboBox, entre otras opciones disponibles.*

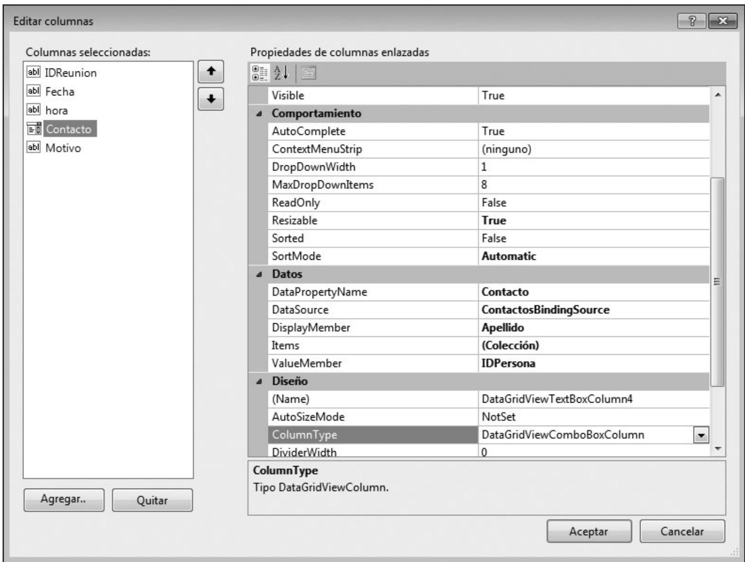


***Figura 21.** De esta manera podremos modificar cualquier celda de DataGridView y convertirla en un control avanzado para mostrar y seleccionar datos de otros orígenes.*

Lo que estamos haciendo a través de este ejercicio es vincular el campo **Contacto**, que es un campo ID relacionado con la tabla **Contactos**, como un **comboBox**, el cual en vez de mostrar números, presentará el contenido del **Apellido** de la tabla **Contactos**. Así, cuando demos de alta un nuevo registro, en lugar de tener que recordar el **ID** del contacto al cual le vinculamos la cita, seleccionaremos el **Apellido**, y Visual Basic se ocupará de obtener el ID correspondiente de la tabla **Contactos** y almacenarlo en el campo **Contacto** de la tabla **Citas**. Por último, antes de probar nuestro proyecto, agregamos al botón **Guardar** del **BindingNavigator** la opción de que refresque los datos desde el objeto **EjemploDataSet**. Para hacerlo, editemos la propiedad **Click()** del botón **CitasBindingNavigatorSaveItem** y agregamos al final del código lo siguiente:

```
frmCitas_Load(sender, e)
```

Esto hará que se cargue otra vez el formulario **frmCitas**, actualizando la información almacenada en la base de datos. Presionamos **F5**, luego hacemos clic en el botón **ABM Citas**, y añadimos directamente un nuevo registro sobre la grilla. En lugar de tener que incluir un ID de contacto, podremos seleccionar el apellido del contacto al cual le aplicaremos dicha cita. Presionamos el botón **Guardar**, salimos de la pantalla y volvemos a ingresar en ella, para verificar que el registro haya quedado almacenado en la fuente de datos.

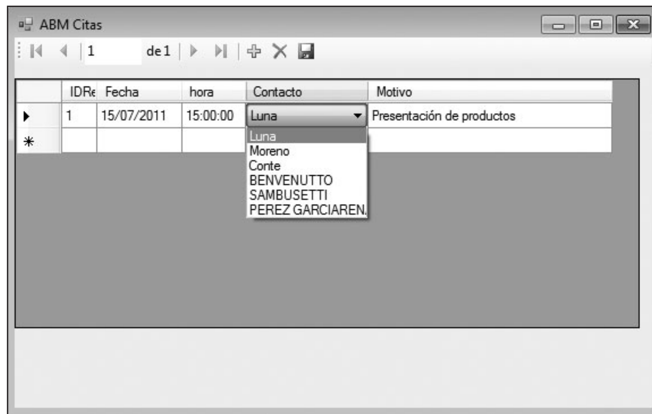


***Figura 22.** Nuestro proyecto de ejemplo funcionando. Veamos cómo la celda **Contacto** fue reemplazada por un **ComboBox** que lista los **contactos disponibles** en la tabla correspondiente.*

Veamos a continuación cuáles son las otras alternativas que VS 2010 nos brinda para que podamos trabajar con registros directamente por código, sin tener que manipular los componentes vistos hasta ahora. Al pie de **frmContactos**, agregamos un botón, llamado **btnAgregarContacto**. Dentro de la declaración de la clase **frmContactos** incluimos la siguiente variable:

```
Public Class frmContactos
Dim cnString as string = "Data
Source=.\AttachDbFilename=C:\Databases\ejemplo.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True"
…
```

Luego agregamos fuera de la clase **frmContactos** la importación de los namespaces que permitirán trabajar mediante código con la base de datos:

```
Imports System.Data.Sql
Imports System.Data.SqlClient

Public Class frmContactos
…
```

El primer código crea una variable del tipo **String** donde se aloja la cadena de conexión para establecer enlace a la base de datos SQL Server 2008 **ejemplo.mdf**. La segunda línea permite trabajar con las clases **System.Data.Sql** y **System.Data.SqlClient** y sus respectivos objetos y clases.

Agregar registros

Para agregar un registro sobre la tabla **Contactos** mediante el uso de código, debemos utilizar una serie de objetos que permitirán la conexión con la fuente de datos y el alta de un nuevo registro. Agregaremos a continuación, en el pie de **frmContactos**, un botón llamado **btnAgregar**, y en cuyo texto figure **Agregar**. Dentro del evento **click()**escribimos el siguiente código:

```
Dim con as new sqlConnection(cnString)
Dim sqlQ as String = "INSERT INTO contactos (Apellido, Nombre, Empresa)
```



USO DE MÓDULOS

El uso de módulos nos da la ventaja de disponer de variables declaradas comunes para todos los Forms utilizados en un proyecto. Para evitar la repetición de código en cada uno, podemos incluir un módulo llamado **Database.vb** y copiar allí la importación de namespaces y la declaración de objetos comunes para operar con bases de datos.

```
VALUES ('Maximo', 'Cartier', 'Mitos Web Design')"
Dim cmd as New SqlCommand(sqlQ, con)
Con.Open()
cmd.ExecuteNonQuery()
Con.Close()
Cmd = Nothing
```

La primera línea establece una conexión nueva con la base de datos **ejemplo.mdf**, utilizando la cadena de conexión especificada anteriormente. **sqlQ** es una variable de tipo **String** donde almacenamos la **sentencia SQL** que permite agregar un nuevo registro a la tabla **contactos**. **cmd** es un **SqlCommand**, y sirve específicamente para ejecutar un comando o sentencia SQL sin devolver resultados en pantalla. **con.Open()** abre la conexión con la base y, a través del método **cmd.ExecuteNonQuery()**, da de alta el nuevo registro. Probemos a continuación si esto realmente funciona, presionando **F5** y luego **btnAgregar**.

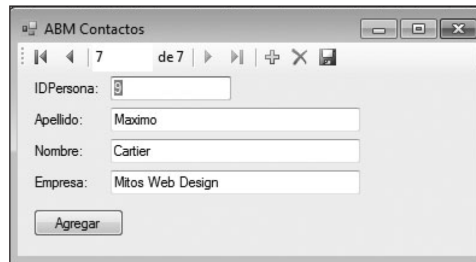


Figura 23. *Nuestro registro fue agregado con éxito, tal como vemos en esta imagen.*

Podemos crear el alta de registros con **textBoxes** y luego especificar su contenido dentro de la consulta SQL de inserción. Modificamos el proyecto ocultando los botones **Agregar**, **Eliminar** y **Guardar** de la barra **ContactosBindingNavigator**; así, de esta manera podemos ordenadamente agregar nuevos registros mediante el uso del código. Solo necesitamos un botón en el cuál tendrá el código encargado de limpiar el contenido de los campos antes de dar de alta un nuevo registro. Agregamos un control **Button** al que llamaremos **btnNuevo**, cuya propiedad **Text** sea **Nuevo**; en el evento **Click()** ponemos el siguiente código:

```
ApellidoTextBox.Text = ""
NombreTextBox.Text = ""
EmpresaTextBox.Text = ""
ApellidoTextBox.Focus()
```

Una vez limpios los campos, reemplazamos los valores por defecto incluidos en la consulta de inserción por la siguiente sentencia SQL:

```
"INSERT INTO contactos (Apellido, Nombre, Empresa) VALUES ('" & Apellido
TextBox.Text & "', '" & NombreTextBox.Text & "', '" & EmpresaTextBox.Text &
 "')"
```

Con estas pocas líneas de código, nuestra aplicación ya es funcional para dar de alta nuevos registros mediante código.

Modificar registros

Modificar registros implica realizar casi los mismos pasos hechos en el alta, pero cambiando la sentencia **SQL** por un **UPDATE** en vez de **INSERT**. Vamos a agregar un nuevo control **Button** llamado **btnModifica**. La propiedad **Text** será **Modificar**, y en el evento **Click()** agregamos el siguiente código:

```
Dim sqlQ as String = "UPDATE contactos SET (Apellido = '" & Apellido
TextBox.Text & "', Nombre = '" & NombreTextBox.Text & "', Empresa = '" &
EmpresaTextBox.Text & "') WHERE IDContacto = " & IDContactoTextBox.Text
Dim cmd as New SqlCommand(sqlQ, con)
Con.Open()
cmd.ExecuteNonQuery()
Con.Close()
Cmd = Nothing
```

La sentencia **UPDATE** permitirá modificar el registro en el cual estamos parados, tomando como base el contenido de cada campo de texto. En la sentencia **SQL** veremos que, antes de guardar la actualización a dicho registro, estamos especificando que tome el valor del campo **IDContacto** para saber qué registro debe actualizar.



OPERACIONES CON REGISTROS CONTROLADAS

En el **Capítulo 6** veremos cómo controlar excepciones y errores que surjan en nuestro programa, y así saber cómo continuar la operación. Todas las sentencias SQL que trabajan con registros siempre deben controlarse mediante **Try Catch**. De esta manera, evitaremos equivocarnos con una sentencia **UPDATE** o **DELETE** propia de SQL.

Eliminar registros

Por último, nos queda ver cómo debemos aplicar la sentencia SQL **DELETE** para eliminar un registro en particular. Agregamos un nuevo botón llamado **btnEliminar**, con la propiedad **Text = Eliminar**, y en su evento **click()** incluimos el siguiente código:

```
Dim sqlQ as String = “DELETE contactos WHERE IDContacto = “ & IDContacto
TextBox.Text
Dim cmd as New SqlCommand(sqlQ, con)
Con.Open()
cmd.ExecuteNonQuery()
Con.Close()
Cmd = Nothing
```

En este caso, especificamos la eliminación de un contacto desde su campo **IDContacto**. Esto nos garantiza que se quitará solamente dicho contacto. Si seleccionamos eliminar un contacto desde su campo **Nombre**, corremos el riesgo de borrar más de un registro si se da la coincidencia de que más de un contacto repita su nombre de pila. Debemos tener sumo cuidado con este detalle. Es por eso que se crearon los campos ID que identifican inequívocamente a un registro.

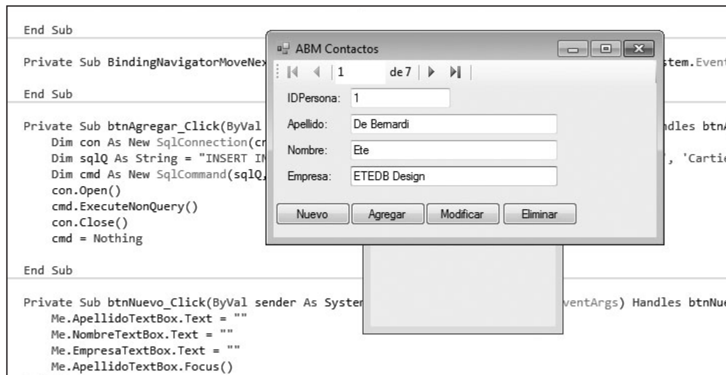


Figura 24. *El ABM de contactos, modificando un registro existente.*

PROYECTO CON BASE DE DATOS: GESTIÓN DE DRUGSTORE

Para cerrar el capítulo de acceso a datos, vamos a desarrollar desde cero un proyecto que integre las operaciones más comunes que podemos realizar sobre un origen de

datos. Este ejemplo consiste en desarrollar un sistema que permita administrar un Drugstore o kiosco de golosinas. Incluirá un **ABM de productos**, un **ABM de categorías** y una ventana de **registro de Ventas**. En principio, crearemos una base de datos nueva en SQL Server, a la que llamaremos **Drugstore**. En dicha base, generamos las siguientes tablas: **Categorias**, **Numeracion**, **Productos**, **Ventas**, y una vista de datos, denominada **vw_ventas**.
Comencemos entonces con la creación de la tabla **categorias**. Su estructura debe ser tal como se describe en la **Tabla 11**.

TABLA CATEGORIAS		
CAMPO	TIPO DE DATOS	OTRAS PROPIEDADES
idcat	BigInt	Aceptar valores Null = False
		Identidad = True
		Incremento de Identidad = 1
		Clave principal = True
nombrecat	NVarChar	Longitud = 50
		Aceptar valores Null = False
activa	Bit	Aceptar valores Null = False
		Valor predeterminado = 1

Tabla 10. *Creación de la tabla Categorias.*

A continuación, en la **Tabla 11** vemos la composición de la tabla **Productos**.

TABLA PRODUCTOS		
CAMPO	TIPO DE DATOS	OTRAS PROPIEDADES
id	BigInt	Aceptar valores Null = False
		Identidad = True
		Incremento de Identidad = 1
		Clave principal = True
categoriaID	BigInt	Aceptar valores Null = False
		Valor por defecto = 0
descripcion	NVarChar	Longitud = 50
		Aceptar valores Null = False
codproveedor	NVarChar	Longitud = 50
		Aceptar valores Null = False
Stock	BigInt	Aceptar valores Null = False
		Valor por defecto = 0
stockminimo	BigInt	Aceptar valores Null = False
		Valor por defecto = 0
preciodecosto	Real	Aceptar valores Null = False
		Valor por defecto = 0

TABLA PRODUCTOS		
CAMPO	TIPO DE DATOS	OTRAS PROPIEDADES
preciodeventa	Real	Aceptar valores Null = False
		Valor por defecto = 0
Activo	Bit	Aceptar valores Null = False
		Valor por defecto = 1

***Tabla 11.** Creación de la tabla **Productos**.*

La tabla **Numeracion** llevará tan solo un campo, que será el que contabilice las nuevas operaciones, como si se tratara de una numeración correlativa de facturas.

TABLA NUMERACIÓN		
Campo	Tipo de datos	Otras propiedades
numticket	BigInt	Aceptar valores Null = False

***Tabla 12.** Creación de la tabla **Numeracion**.*

Por último, la tabla **Ventas** será la encargada de almacenar el historial de cada venta registrada en el drugstore. En la **Tabla 13** vemos los campos que contendrá.

TABLA VENTAS		
CAMPO	TIPO DE DATOS	OTRAS PROPIEDADES
Idm	BigInt	Aceptar valores Null = False
		Identidad = True
		Incremento de Identidad = 1
Fecha	Datetime	
nroventa	BigInt	Aceptar valores Null = False
Id	BigInt	Aceptar valores Null = False
cantidad	BigInt	Aceptar valores Null = False
preciounitario	Real	Aceptar valores Null = False
subtotal	Real	Aceptar valores Null = False

***Tabla 13.** Creación de la tabla **Ventas**.*

Luego nos queda crear la vista, que llamaremos **vw_ventas**. Para esto, desde el mismo **SQL Server Management Studio**, presionamos el botón derecho del mouse sobre la carpeta **Vistas** y, del menú contextual, elegimos **Nueva vista...** Agregamos a continuación las tablas **Ventas** y **Productos**. De la primera seleccionamos los campos **nroventa**, **id**, **cantidad**, **preciounitario** y **subtotal**; y de la segunda, solamente **descripcion**. Establecemos una relación entre el campo **ventas.id** y **productos.id**, la cual se basará en **Seleccionar todas las filas desde ventas**. Por último, guardamos la vista con el nombre mencionado. La estructura debe quedar como se observa en la **Figura 25**.

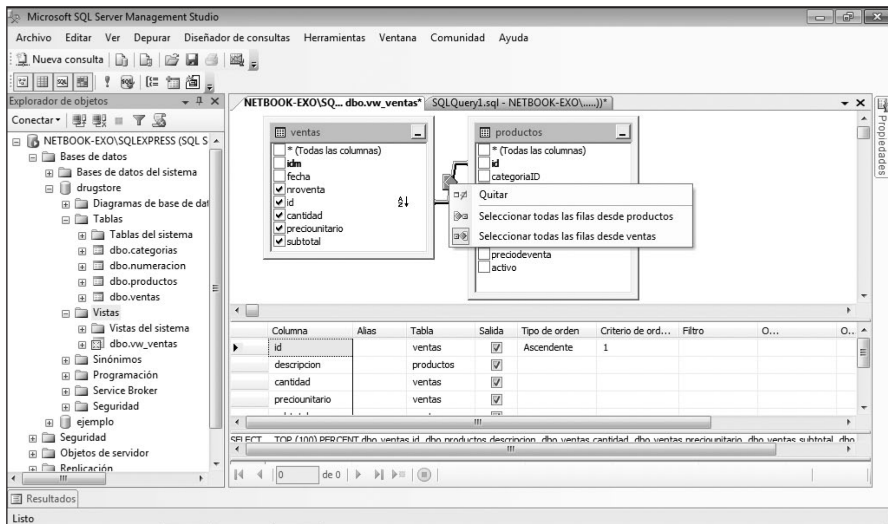


Figura 25. *Creación de una vista en* ***SQL Server Management Studio.***

A continuación vamos a generar un nuevo proyecto del tipo **Windows Forms Application**, llamado **Drugstore**. Al **form1** lo renombramos como **frmMain** y le agregamos cuatro controles **Button**. El primero se llama **btnABMCat** y en su propiedad **Text** ponemos **ABM Categorías**. El segundo se denomina **btnProductos**, y su propiedad **Text** será **ABM Productos**. El tercero es **btnSTKMinimo** y su propiedad **Text** será **Stock Mínimo**. Al cuarto botón lo llamamos **btnVenta** y su propiedad **Text** será Ventas. Esta será la pantalla principal de la aplicación, desde donde ingresaremos a los distintos módulos que componen el sistema para administrar los productos, categorías, stock mínimo y ventas.



Figura 26. ***El diseño de la pantalla inicial debe quedar como se muestra en esta figura.***

En el siguiente paso, crearemos el formulario de administración de las categorías de productos. Dicho **form** incluirá un **DataGridView** enlazado a datos, el cual nos

permitirá, a su vez, crear el **DataSet** que incluirá todos los objetos creados en la base de datos. Agregamos un nuevo Form, lo llamamos **frmABMCategorias**. Arrastramos a él un control **DataGridView**, en donde seleccionamos un nuevo origen de datos. Seguimos el asistente de nuevo origen de datos, seleccionando como base de datos SQL Server el archivo **drugstore.mdf**. Luego seleccionamos todas las tablas y vistas de dicha base, y guardamos el **DataSet** con el nombre **DrugstoreDataSet**. El **BindingSource** vinculado a categorías será **CategoriasBindingSource**. Por último, editamos las columnas de **DataGridView1** para ajustar sus propiedades como lo indica la **Tabla 14**.

CAMPO	VALORES
Codigo	Header Text = Código
	DataGridViewCellStyle.Alignment = MiddleRight
	Width = 50
nombrecat	Header Text = Nombre de categoría
	DataGridViewCellStyle.Alignment = MiddleLeft
	Width = 270
Activa	Header Text = Activa
	Width = 40

***Tabla 15.** Ajuste de propiedades de DataGridView1 en ABM **Categorías**.*

En las tareas de **DataGridView1** dejamos marcadas las opciones **Agregar**, **Edición**, **Eliminación** y **Reordenación** de columnas. Luego agregamos dos controles **Button** al **frmABMCategorias**. Uno se llamará **btnUpdate** y como propiedad **Text = Actualizar cambios**, el otro **Button** será **btnCerrar** y como propiedad **Text = Cerrar**. Por último agregamos el siguiente código a **btnUpdate_Click():**

```
CategoriasBindingSource.EndEdit()
CategoriasTableAdapter.Update(DrugstoreDataSet.categorias)
MessageBox.Show("Se ha actualizado la tabla CATEGORIAS.")
```

En **btnCerra**r incluimos el código para cerrar dicho **Form**.



VISTA EN GRILLA O INDIVIDUAL

Siempre existe el dilema de si crear o no un form con conexión a datos a través de un **DataGrid-View** o vista individual de registros. Por lo general, conviene la primera opción si es que vamos a trabajar solamente con una tabla. La segunda modalidad suele utilizarse para tablas con muchos registros o vistas que requieren la interacción con diversas tablas.

```
Me.Close()
```

A continuación, agregamos un nuevo **Form**, llamado **frmABMProductos**. Como ya tenemos creado el **DataSet** con la relación a todas las tablas y vistas de la base de datos, aprovechamos la pestaña **Orígenes de datos** y, desde allí, agregamos la tabla a dicho Form con la modalidad **Vista Detalles**. Se crearán todos los campos de manera automática, al igual que el **ProductosBindingNavigator**, que nos permitirá desplazar entre los registros existentes. Esta vista la elegimos porque la cantidad de campos es significativamente superior a la de la tabla Categorías, y visualizarlo así será más cómodo. Una vez creados todos los campos y controles, agregamos un control **ComboBox**, que nos permitirá obtener, de manera automática, una lista de las categorías disponibles cuando demos de alta un producto nuevo. Este ítem seleccionado impactará directamente en el campo **CategoriaIDTextBox** de la tabla **Productos**. Al agregar el comboBox, desde sus tareas configuramos el origen de datos y demás propiedades necesarias, como lo indica la **Tabla 15**.

COMBOBOX	
Propiedad	Valor
Origen de datos	CategoriasBindingSource
Mostrar miembro	Nombrecat
Miembro de valor	Idcat
Valor seleccionado	categoriaID

Tabla 15. *Configuración del enlace a datos de comboBox1 para mostrar los datos de la tabla Categorías.*

Para hacer efectivo el uso del **ComboBox1**, podemos probarlo y luego ocultar el campo **CategoriaIDTextBox**, ya que no lo utilizaremos para nada. El resultado de la creación de dicho **Form** debe quedar tal como lo indica la **Figura 27**.

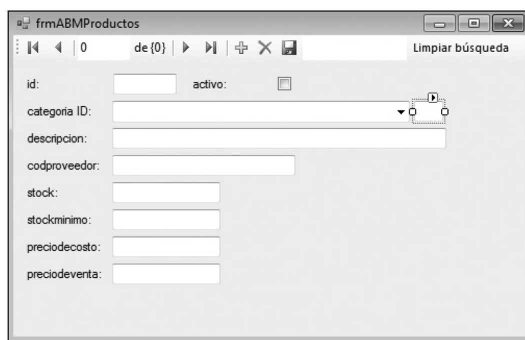


Figura 27. *El diseño final de la pantalla ABM Productos, con la vinculación a los datos de la tabla Categorías mediante ComboBox1.*

Como el módulo **ABM productos** puede tener una cantidad importante de registros, debemos incluir una opción de filtro por el nombre o parte del nombre de un producto. En la barra de herramientas **ProductosBindingNavigator** agregamos algunos objetos más. El primero será un **textBox** llamado **txtBuscoDesc**, y luego, un **Button** denominado **ProductosLimpiarBindingItem**, cuyo valor **Text = Limpiar búsqueda**. Dentro de la propiedad **Text_Changed()** de **txtBuscoDesc** incluimos el siguiente código:

```
If trim(txtBuscoDesc.Text) = “” then
     Me.ProductosBindingSource.RemoveFilter()
Else
     Me.ProductosBindingSource.Filter = “descripcion LIKE ‘*” &
trim(txtBuscoDesc.Text) & “*’”
endif
```

En el botón **LimpiarBindingItem_Click()** agregamos el siguiente código:

```
Me.ProductosBindingSource.RemoveFilter()
Me.txtBuscoDesc.Text = “”
```

El texto ingresado en **txtBuscoDesc** aplicará inmediatamente un filtro con las coincidencias encontradas dentro de los registros de la tabla **Productos**. Al utilizar la cláusula **LIKE** al igual que en una sentencia SQL con asteriscos al inicio y al final, se buscará(n) la(s) coincidencia(s) en cualquier parte del campo descripción.

Figura 28. *El filtro aplicado sobre 34 registros nos da como coincidencia 5 que incluyen "pastill" en su descripción.*

Luego creamos un nuevo **Form**, al cual llamaremos **frmSTKM**, y cuya propiedad **Text** será **Consulta de Stock Mínimo**. Su finalidad es saber cuándo los productos del **drugstore** sobrepasan un stock mínimo identificado. Esto se conoce como punto de nuevo

pedido, para prevenir que el negocio que utiliza dicho software se quede sin stock de mercadería. Agregamos un control **DataGridView**, enlazamos los datos de la grilla a **ProductosBindingSource** y personalicemos la vista de los campos mostrados. Para esto, vamos al menú de **Tareas** y elegimos **Editar columnas...**. Luego, seleccionamos las columnas **id**, **descripcion**, **codproveedor**, **stock** y **stockminimo**. Como será una ventana de solo consulta, desmarcamos las acciones de **agregar**, **edición**, **eliminación y reordenación de columnas**. Agregamos una **nueva consulta** de datos denominada **FillBySTKMinimo()**, dentro de la cual especificamos la siguiente sentencia SQL:

```
SELECT id, descripcion, stock, stockminimo FROM productos WHERE (stock <=
stockminimo)
```

Esto permitirá conocer qué productos tienen un stock por debajo del punto de pedido.

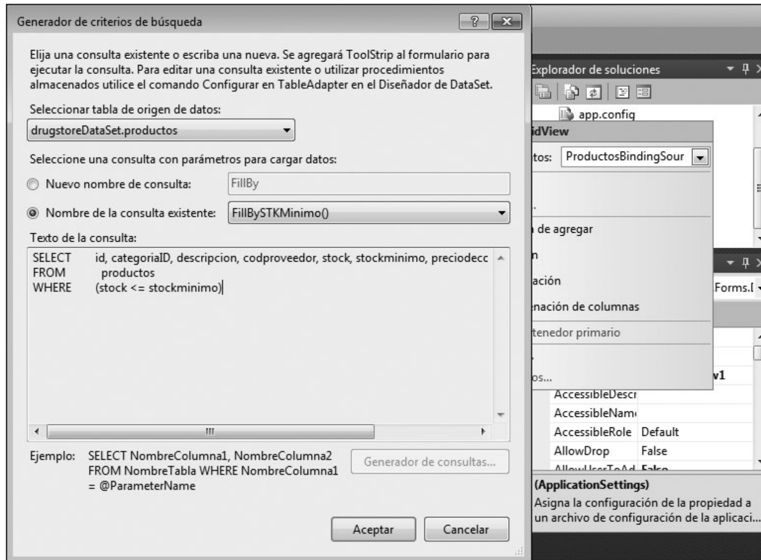


Figura 29. *La consulta nos permite ver los productos que están por debajo de su punto mínimo de pedido.*



MÚLTIPLES CONSULTAS

Cada objeto **BindingSource** nos permite crear múltiples consultas ingresando un criterio distinto para cada una. Esto hará que el resultado pueda visualizarse mucho más rápido aplicando la invocación **nombredelBindingSource.FillByNombreDeConsulta()**. Así evitaremos efectuar reiteradas consultas a un motor de datos si este ya tiene predefinidos los resultados en **caché**.

Agreguemos un nuevo Form llamado **frmVenta**, titulado **Venta al público**, y luego los siguiente controles:

FRMVENTA		
CONTROL	NOMBRE	PROPIEDADES
Button	btnNuevaVenta	Text = Nueva venta
	btnCancelar	Text = Cancelar venta
		Enabled = False
	btnGrabar	Text = Grabar venta
		Enabled = False
Label	Label5	Text = Venta:
	lblVenta	Text = 0
		TextAlign = MiddleRight
textBox	txtSelecciono	
Button	btnSelecciona	Text = Seleccionar
Label	lblDesc	Text = Descripción del producto
	lblStk	Text = Stock actual
	lblImporte	Text = Importe unitario
	lblSubt	Text = Sub total
textBox	txtCantidad	
DataGridView	gridProd	Origen de datos = ProductosBindingSource
		Deshabilitar las propiedades de acciones.
		Visible = False
		Campos a visualizar: ID, descripcion, Stock, Importe, Activo
Button	btnAgregar	Agregar
DataGridView	gridVenta	Origen de datos = vwVentasBindingSource
		Deshabilitar las propiedades de acciones menos Eliminación.
		Campos a visualizar: ID, descripcion, cantidad, Importe, Subtotal, nroventa.
		Campo ID.Width = 0
		Campo nroVenta.Width = 0
Label	lblTotal	Text = 0.00

Tabla 16. *Creación del **frmVenta** con sus controles y propiedades.*



FUNCIONES DE SQL SERVER

Server permite crear funciones propias a través del lenguaje TRANSACT SQL, las cuales aceptan parámetros y pueden realizar determinadas acciones sobre una o más tablas. Son utilizadas generalmente cuando deben realizarse cálculos complejos, los cuales conviene que sean procesados en el motor de la base de datos.

El diseño final de este Form debe quedar como muestra la **Figura 30**.

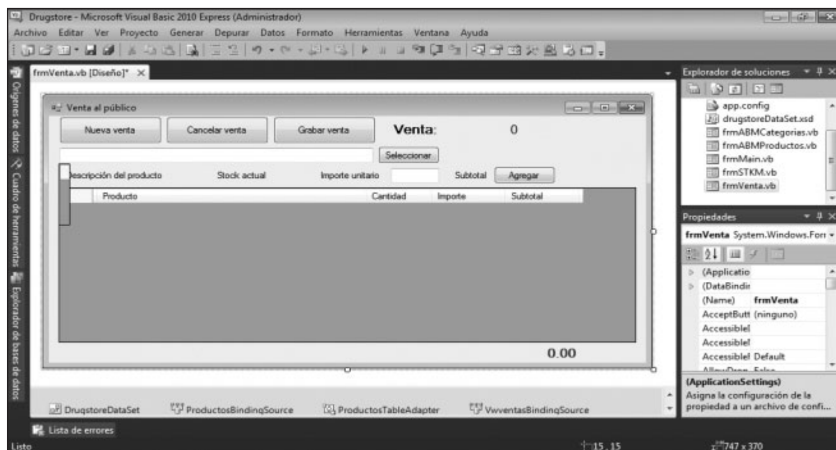


***Figura 30.** Distribución de los controles del formulario de ventas. El tamaño de **gridProd** se manejará mediante código, por lo que se muestra compactado en pantalla.*

Acto seguido, agregamos el código en el interior de cada control para que el proyecto quede funcional. Añadimos al principio de la clase **frmVenta** la incorporación de los **namespaces** de datos y realizamos también la declaración de la variable con la cadena de conexión a la base de datos:

```
Imports System.Data.Sql
Imports System.Data.SqlClient

Public Class frmVenta
Dim cnString As String = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\data
bases\drugstore.mdf;Integrated Security=True;Connect Timeout=30;User
Instance=True"
…
```

Creamos una función que grabará un nuevo número cuando se confirme una venta:

```
    Function graboNuevoNro() As Boolean
        Dim con As New SqlConnection(cnString)
        Dim sqlQ As String = "UPDATE numeracion SET numticket = (numticket
 + 1)"
        Dim cmd As New SqlCommand(sqlQ, con)
        graboNuevoNro = False
```

```
        con.Open()
        cmd.ExecuteNonQuery()
        graboNuevoNro = True
End Function
```

Generamos a continuación una función que bloqueará de manera automática los botones de acciones de **frmVenta**:

```
    Sub cambioBotones()
        btnNuevaVenta.Enabled = Not btnNuevaVenta.Enabled
        btnCancelar.Enabled = Not btnCancelar.Enabled
        btnGrabar.Enabled = Not btnGrabar.Enabled
    End Sub
```

Creamos una función que nos traerá de la base de datos el último número utilizado y le sumará 1 para crear una nueva venta:

```
    Function traigoUltimoNro() As String
        Dim dbConnection As New SqlConnection(cnString)
        Dim sqlQ As String = “SELECT numticket FROM numeracion”
        Dim Reader As SqlDataReader
        Dim cmd As New SqlCommand(sqlQ, dbConnection)
        traigoUltimoNro = -1
            dbConnection.Open()
            cmd.Connection = dbConnection
            cmd.CommandText = sqlQ
            Reader = cmd.ExecuteReader
            Reader.Read()
            traigoUltimoNro = FormatNumber(Reader.Item(0).ToString, 2) + 1
            Reader.Close()
    End Function
```

Este es el código para iniciar el módulo con una nueva venta:

```
Private Sub btnNuevaVenta_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnNuevaVenta.Click
        lblVenta.Text = traigoUltimoNro()
```

```
        Me.VwventasBindingSource.Filter = "nroventa = " &
CSng(lblVenta.Text)
        txtSelecciono.Focus()
        cambioBotones()
End Sub
```

En el evento **Form_Load()** se debe especificar el ocultamiento de algunos controles y la eliminación de los filtros de productos. También filtramos el **DataGridView gridProd**, para que no muestre ningún registro de entrada:

```
    Private Sub frmVenta_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Me.Vw_ventasTableAdapter.Fill(Me.DrugstoreDataSet.vw_ventas)
Me.ProductosTableAdapter.Fill(Me.DrugstoreDataSet.productos)
        Me.VwventasBindingSource.Filter = "nroventa = 0"
    End Sub
```

Al escribir parte del nombre de algún producto, se activa la grilla de productos y se muestra su contenido con los resultados hallados:

```
Private Sub txtSelecciono_TextChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles txtSelecciono.TextChanged
        If Trim(txtSelecciono.Text) = "" Then
            Me.ProductosBindingSource.RemoveFilter()
            Me.gridProd.Visible = False
        Else
            Me.ProductosBindingSource.Filter = "descripcion LIKE '*" &
Trim(txtSelecciono.Text) & "%'"
            Me.gridProd.Visible = True
```



CIFRADO EN SQL SERVER

SQL Server, dentro de su lenguaje TRANSACT SQL, incluye también diversas funciones de cifrado, que permiten cifrar, descifrar, firmar digitalmente y validar firmas digitales realizadas con este lenguaje. Las funciones de encriptación son útiles cuando hay que cifrar contraseñas en un motor de base de datos.

```
            Me.gridProd.Width = 608
        End If
End Sub
```

Seleccionamos el producto deseado de la grilla de productos, y luego, presionamos el botón **Selecciona** para que lo cargue como producto seleccionado y permita especificar la cantidad que se está vendiendo:

```
Private Sub btnSelecciona_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSelecciona.Click
        With gridProd
            If gridProd.Rows.Count = 0 Then Exit Sub
            lblDesc.Tag = .CurrentRow.Cells(0).Value
            lblDesc.Text = .CurrentRow.Cells(1).Value
            lblStk.Text = Format(.CurrentRow.Cells(2).Value, 2)
            lblImporte.Text = Format(.CurrentRow.Cells(3).Value, “c”)
        End With
        txtSelecciono.Text = “”
        txtCantidad.Focus()
End Sub
```

Al ingresar la cantidad de productos que se están vendiendo, el sistema calcula de manera automática cuál es el precio total de dicha cantidad:

```
Private Sub txtCantidad_TextChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles txtCantidad.TextChanged
        If Trim(txtCantidad.Text) = “” Then lblSubt.Text = 0 : Exit Sub
        If IsNumeric(Trim(txtCantidad.Text)) Then
            lblSubt.Text = Format(CSng(txtCantidad.Text) * CSng(lblIm
```



LENGUAJE T-SQL DESDE VB

Al igual que la mayoría de los lenguajes que permite el acceso a datos mediante sentencias SQL, Visual Basic puede aprovechar las ventajas de este lenguaje de transacciones y realizar operaciones como la creación de una base de datos, tablas, vistas y otros objetos dentro de un SQL Server, o cualquier otro motor que estemos utilizando.

```
porte.Text), "c")
        End If
End Sub
```

Cuando agregamos productos a la venta actual, se debe calcular el total de dicha venta, que será informado al cliente. Para esto, necesitamos un procedimiento que calcule el total del monto hasta el momento:

```
Sub calculoTotal()
    Dim cn As New SqlConnection(cnString)
    Dim sqlQ As String = "SELECT sum(subtotal) AS Total FROM ventas WHERE
nroventa = " & CSng(lblVenta.Text)
    Dim cmd As New SqlCommand(sqlq, cn)
    Dim Reader As SqlDataReader
    cn.Open()
    cmd.Connection = cn
    cmd.CommandText = sqlQ
    Reader = cmd.ExecuteReader
    Reader.Read()
    Dim d As Double = CDbl(Reader.Item(0))
    lblTotal.Text = Format(d, "c")
    Reader.Close()
End Sub
```

Una vez definida la cantidad de lo que se está vendiendo, lo vinculamos a la venta actual:

```
Private Sub btnAgregar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAgregar.Click
        Dim dbConnection As New SqlConnection(cnString)
        Dim sqlInsert As String = "INSERT INTO ventas (nroventa, id,
cantidad, preciounitario, subtotal) "
        Dim sqlValores As String = "VALUES(" & CSng(lblVenta.Text) & ", "
& lblDesc.Tag & ", " & txtCantidad.Text & ", " & lblImporte.Text & ", " &
lblSubt.Text & ")"
        Dim cmd As New SqlCommand(sqlInsert + sqlValores, dbConnection)
            dbConnection.Open()
            cmd.ExecuteNonQuery()
            Me.Vw_ventasTableAdapter.Fill(Me.DrugstoreDataSet.vw_ventas)
```

```
            Me.VwventasBindingSource.Filter = "nroventa = " &
CSng(lblVenta.Text)
            cmd = Nothing
            dbConnection.Close()
            lblDesc.Text = "Descripción del producto"
            lblStk.Text = "Stock actual"
            lblImporte.Text = "Importe unitario"
            txtCantidad.Text = ""
            lblSubt.Text = "Subtotal"
            calculoTotal()
End Sub
```

Debemos confirmar la venta para sumar el número de venta en la base de datos:

```
Private Sub btnGrabar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGrabar.Click
        If graboNuevoNro() Then
            cambioBotones()
            LimpioCampos()
            Dim msg As String = "Se ha grabado correctamente la venta."
            MessageBox.Show(msg, "Nueva venta", MessageBoxButtons.OK,
MessageBoxIcon.Information)
        End If
End Sub
```

Falta limpiar los campos y dejar la pantalla lista para poder realizar una nueva venta:

```
Sub LimpioCampos()
    Me.VwventasBindingSource.Filter = "nroventa = 0"
    lblVenta.Text = "0"
    txtSelecciono.Text = ""
    lblTotal.Text = "0.00"
    lblDesc.Text = "Descripción del producto"
    lblStk.Text = "Stock actual"
    lblImporte.Text = "Importe unitario"
    txtCantidad.Text = ""
    lblSubt.Text = "Subtotal"
End Sub
```

Puede suceder que la venta se cancele, con lo cual tenemos que deshacer los productos agregados y liberar el número de venta para una próxima:

```
Private Sub btnCancelar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCancelar.Click
        Dim cn As New SqlConnection(cnString)
        Dim sqlQ As String = "DELETE FROM ventas WHERE nroventa = " &
CLng(lblVenta.Text)
        Dim cmd As New SqlCommand(sqlQ, cn)
        If MessageBox.Show("Está seguro de eliminar la venta actual?",
"Cancelar venta", MessageBoxButtons.YesNo, MessageBoxIcon.Question) = vbYes
Then
            cn.Open()
            cmd.ExecuteNonQuery()
            LimpioCampos()
            cambioBotones()
        End If
    End Sub
End Class
```

Con este código, nuestro proyecto ya es funcional. Solo resta agregar el código que llame a cada **Form** desde **frmMain**, y podremos probar el sistema:

```
btnProductos_Click(…)
frmABMProductos.Show()
End Sub

btnSTKMinimo_Click(…)
frmSTKM.ShowDialog()
End Sub
```



PLANIFICACION DE UN SISTEMA

Tengamos en cuenta que un sistema requiere de una planificación detallada, a través de diagramas de flujo y declaración de muchas clases importantes que serán reutilizadas sin repetición de código a lo largo de todo el proyecto. Conocer UML es importante en esta profesión.

```
btnVenta_Click(…)
frmVenta.ShowDialog()
End Sub
```

Presionamos **F5** y veremos la funcionalidad del sistema.

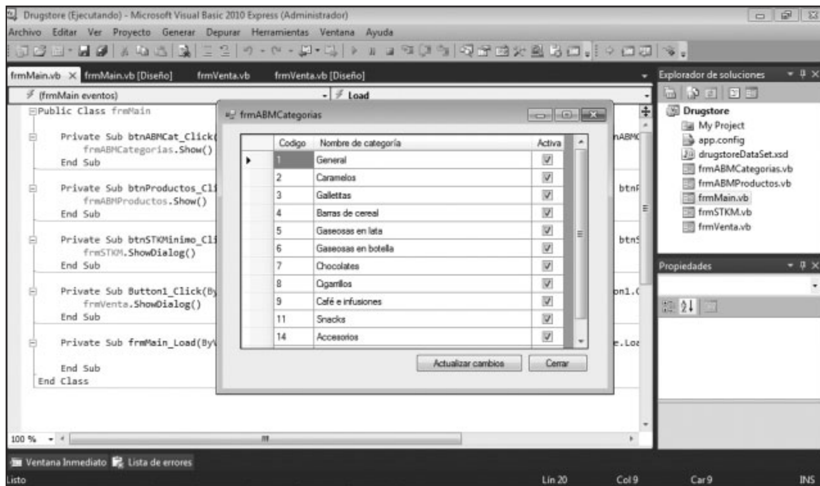


Figura 31. *Agregamos algunas categorías en principio, para luego añadir productos vinculados a ellas.*

En el alta de productos, podemos apreciar en funcionamiento el **ComboBox** responsable de listar las categorías existentes.

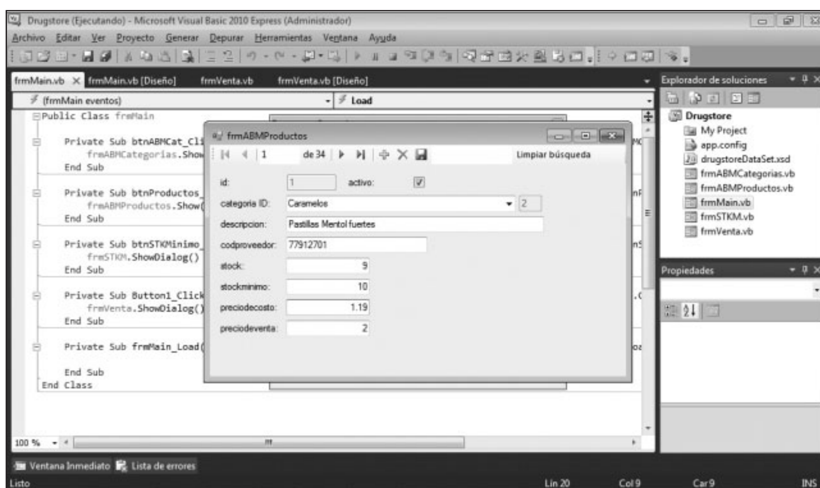


Figura 32. *Debemos recordar que cada producto que agreguemos necesita tener el precio de compra y el de venta por separado.*

Podemos personalizar los colores de cada **DataGridView** para indicar con ellos las diferencias entre columnas, y así visualizar cómodamente los resultados en pantalla.

***Figura 33.** La columna Stock muestra el stock real, mientras que la columna Mínimo muestra el punto de pedido para cada producto.*

El módulo **frmVenta** funciona a la perfección con dos controles **DataGridView**. También muestra en tiempo real el total de la venta realizada hasta el momento.

***Figura 34.** Los productos listados en la grilla son solamente aquellos que tengan stock y estén activos para la venta.*

Siempre es conveniente notificar al usuario del sistema mediante un mensaje si la venta se pudo grabar con éxito o si ocurrió algún error durante la transacción. Y todo mensaje de notificación generado por el sistema debe ser lo más claro posible.

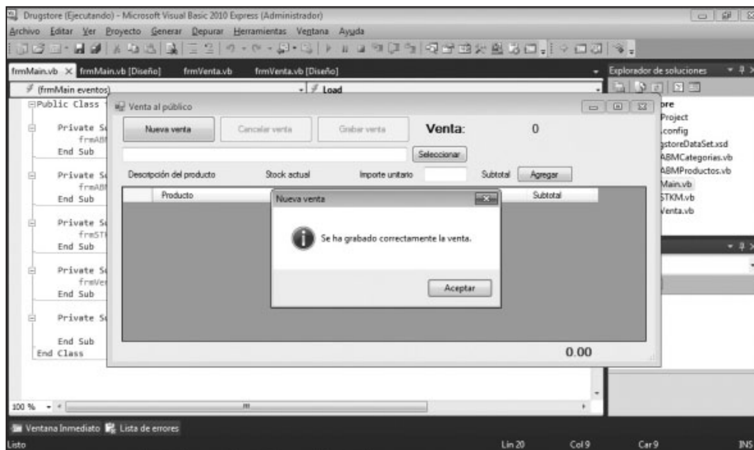


Figura 35. *Finalización con éxito de la venta realizada por sistema.*

Al iniciar una nueva venta, podemos apreciar que la numeración es correlativa y distinta de la anterior, gracias a la función creada especialmente para actualizar el número de venta de manera automática.

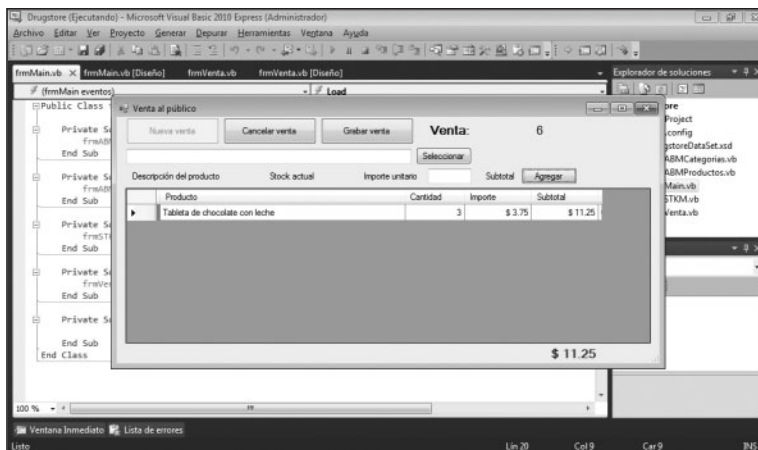


Figura 36. *Iniciando una nueva venta con su nuevo número de operación.*

RESUMEN

Hasta aquí hemos logrado repasar cómo manejarnos correctamente con un motor de base de datos potente y eficaz como es SQL Server, y cómo acceder a datos a través de Visual Basic 2010. Hicimos un repaso por la sintaxis SQL para consulta y operaciones con registros, y creamos un sistema de ventas casi completo donde implementamos varias modalidades de acceso a datos y operaciones con registros.



ACTIVIDADES

EJERCICIOS PRÁCTICOS

1. Cree una función que recorra cada registro agregado en el módulo de ventas, dentro de una nueva venta, para que, al grabarla, descuente del stock de ese producto la cantidad que se está vendiendo.

2. Agregue una función que, al ingresar en la aplicación, si es que existen productos con stock por debajo del punto de pedido indicado, permita poner en negrita el texto del botón **Stock Mínimo**.

3. Incorpore una función a la pantalla de productos que permita incrementar de forma masiva el precio de venta de todos los productos, especificando un porcentaje por el usuario del sistema.

4. Agregue una función similar a la anterior, en el módulo de **Categorías**, que permita incrementar en porcentaje los precios de todos los productos vinculados a la categoría por la cual está navegando en ese momento.

5. Modifique el módulo de ventas para que permita buscar también un producto por su código de proveedor, en lugar de permitir sólo por su nombre.

6. Investigue qué otra base de datos puede abrir con Visual Basic 2010 de manera nativa.

7. Investigue cómo crear con Transact SQL un Stored Procedure en SQL Server.

8. Intente reemplazar uno de los **SELECT** utilizados en el ejercicio práctico de este capítulo por un Stored Procedure.

9. Busque en Internet al menos dos diferencias entre el lenguaje SQL utilizado en el motor JET de Access y SQL Server.

10. Investigue de qué manera almacenar una imagen en un campo de una base de datos SQL Server, e intente crear un proyecto que le deje seleccionar una imagen del disco y guardarla. Luego realice un evento que permita recuperar la imagen desde la BBDD.

Depuración y manejo de errores

Depurar el código es esencial en el desarrollo de todo sistema. Esto nos permite encontrar errores tanto en el código como en la lógica de la aplicación. Por eso, este capítulo nos introduce en el manejo de errores en Visual Basic 2010. Estudiaremos las sentencias TRY, CATCH y FINALLY, conoceremos las excepciones, veremos cómo continuar el flujo de la aplicación ante un error y repasaremos las características de depuración.

CAMBIOS IMPORTANTES EN EL MANEJO DE FLUJO

No importa cuán bueno sea uno como programador y cuán bueno sea nuestro código al escribirlo; podemos tener cientos de controles que eviten llegar a un error en la aplicación, pero como todo ser humano, en algún momento de nuestra vida cometemos equivocaciones, y seguramente estos errores o descuidos se verán también reflejados en el software. Todo programa es estructurado por más que la orientación de su plataforma no requiera que así lo sea. Cuando creamos una **función** o un **método**, **o establecemos una propiedad a una variable** u **objeto**, estamos estructurando la manera en la que ese objeto debe trabajar. Si esa estructura de trabajo que indicamos es desviada de su curso por algún motivo que no tenemos en cuenta, esto producirá un error en el software. Quienes hayan programado en alguna versión antigua de Visual Basic, desde que este lenguaje fue orientado a objetos, recordarán que la única manera de controlar los errores de las aplicaciones era a través de las sentencias **`On error Goto`**, **`On error Resume Next`** u **`On error Goto 0`**. Desde la llegada de la plataforma .NET, Visual Basic logró muchas mejoras y cambios en el control de errores en tiempo de ejecución. Principalmente, dentro de la tecnología **Intellisense** de la suite Visual Studio .NET, se incluyó la posibilidad de detectar errores al momento de escribir el código. Esto era algo que no ocurría hasta que ejecutábamos el proyecto para probarlo, hasta que lo compilábamos o hasta que lo testeábamos ya funcionando. En la actualidad, Visual Basic incluye un detector de errores desde el mismo momento en que escribimos el código. Esto ya reduce en un porcentaje importante la posibilidad de cometer equivocaciones dentro de la aplicación. Repasaremos a continuación algunos ejemplos más claros antes de continuar viendo en profundidad este tema.

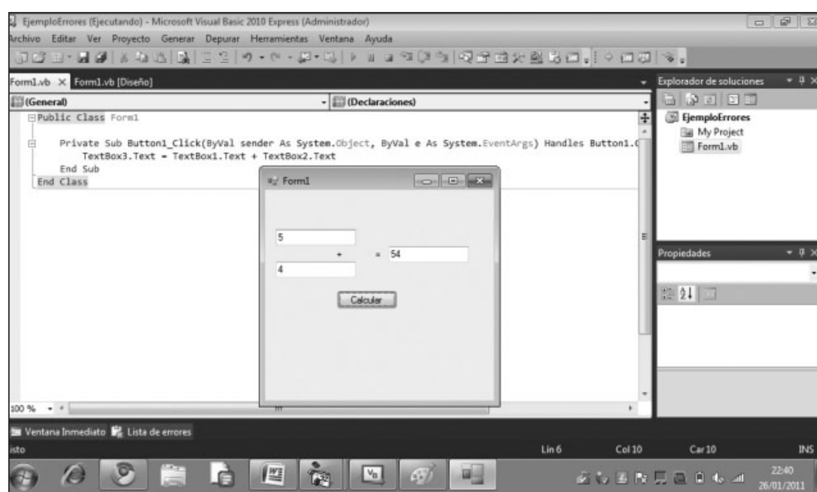


Figura 1. *Ejemplo de una típica equivocación que puede ocurrirnos como programadores novatos en cualquier lenguaje.*

En la **Figura 1** podemos apreciar un clásico error que cometemos cuando comenzamos a programar en los lenguajes, hasta que logramos conocer los conceptos de tipos de datos para la declaración de variables. Lo que ocurre en este ejemplo no es un error grave de programación, sino de concepto. En ningún momento indicamos que los valores contenidos en **textbox1** y **textbox2** son del tipo numérico, con lo cual el lenguaje de programación, ciento por ciento orientado a objetos, interpreta que lo ingresado en una caja de texto es una cadena de caracteres, por lo que debe concatenarlos. Para esto, debemos trasladar el valor de cada caja de texto a una variable del tipo numérica, y recién entonces se podrá realizar la suma, dado que el intérprete comprenderá que, cuando son números, debe sumarlos en vez de concatenarlos. Veamos qué ocurre si declaramos dos variables numéricas para poder realizar la operación.

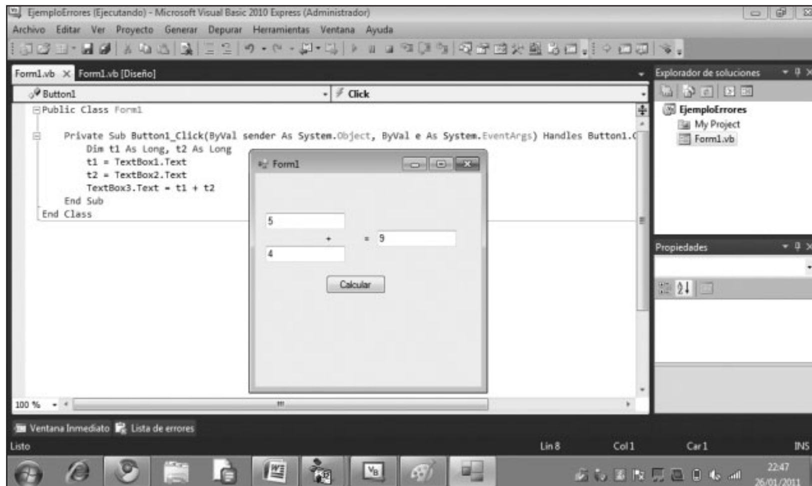


***Figura 2.** Corrigiendo nuestra primera equivocación, ya tenemos el resultado que buscamos.*

Como vemos en la **Figura 2**, ahora sí podemos obtener el resultado que estamos buscando. A simple vista, mirando el código, nuestro problema está solucionado, pero si miramos el código con detenimiento y analizamos distintas variables, nos



HERENCIA EN EL MANEJO DE ERRORES

Hasta el momento de la masificación de los lenguajes de programación orientados a objetos, el manejo de errores en las versiones anteriores se llevaba a cabo a través de un ErrorHandler o manejador de errores. Estos nunca eran ciento por ciento infalibles, y a la larga, terminaban por colapsar la aplicación y cerrándola sin poder controlarla.

daremos cuenta de que sigue habiendo errores en el algoritmo que no están siendo contemplados, y ante una equivocación humana, esto fallará otra vez.
Ingresemos en la primera casilla la palabra cinco, y en la siguiente, la palabra cuatro o, directamente, el número 4, y veamos qué ocurre.

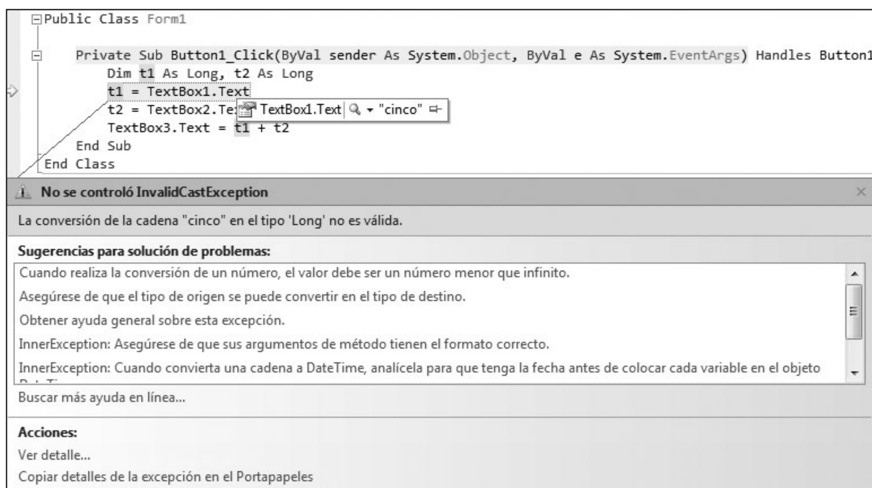


Figura 3. *Este tipo de error no se ve a simple vista y, generalmente, es de los que más aparecen cuando no hay control de errores.*

Nuestra aplicación podría haber sido compilada sin tener en cuenta lo que ocurre en la **Figura 3**, y al ser distribuida, podría haber ocurrido esto sin contemplarlo. Este tipo de error en tiempo de ejecución cerrará el programa sin posibilidad de continuar utilizándolo. En este tipo de ejemplo no ocurre nada, pero si esto no se contempla en un sistema importante, o de misión crítica, que lleve adelante la contabilidad de una empresa, seguramente que quienes la dirigen querrán prescindir de nuestros servicios lo antes posible.

MANEJO DE ERRORES EN TIEMPO DE EJECUCIÓN

La limitación del manejo de errores en tiempo de ejecución aún existe. A pesar de la gran evolución que ha tenido la plataforma .NET, muchas veces, durante el desarrollo de grandes soluciones de software, se requiere utilizar herramientas de terceros. Estas suelen ser librerías del tipo **.DLL**, las cuales, seguramente, fueron hechas con **Visual C++**. En este caso, el control de errores ya no depende de nuestra aplicación, sino del desarrollo que haya hecho el tercero en el control de calidad de su producto, lo cual nos limita en determinados casos para poder dar una respuesta clara y concisa al usuario sobre lo ocurrido.

Qué son las excepciones

Visual Basic 2010, a través del framework .NET, implementó una normalización a través de un único método de provocar y atrapar los errores que ocurren dentro de los algoritmos. Esto fue bautizado como excepciones. Este tipo de control fue heredado de lenguajes como C++ y Java, que utilizan el mismo concepto para manejar errores en tiempo de ejecución de la aplicación o librería. Podemos decir que una excepción es una condición que no es esperada durante la ejecución de la aplicación, o cuando el código se está ejecutando dentro del framework .NET. Al ocurrir esto, se entiendo que el código inicia una excepción dentro de la rutina que venía procesando, con lo cual espera que haya una contemplación de código que sea capaz de capturar este error. Las excepciones pueden ser capturadas dentro del procedimiento o función donde se produce el error. Si esto no sucede, la excepción se comunica al llamador, que será el encargado de tratar de reintentar la operación que condujo el flujo del programa hasta ese punto. Si el **llamador** no es capaz de frenar la excepción, esta última invocará al **llamador del llamador**. La aplicación sigue produciendo saltos dentro de su código tratando de localizar un procedimiento que permita frenar el error. En caso de que esto no ocurra, el actor de la aplicación será notificado mediante un cuadro de diálogo con la descripción de dicho error y con la sugerencia de depurarlo a través de alguna herramienta instalada en la computadora. El capturador de errores del framework .NET espera un valor como resultado para saber qué debe continuar haciendo. Si el valor que recibe es **0**, aparecerá un cuadro de diálogo con la descripción técnica del error; si es **1**, la aplicación se cerrará; y si es **2**, se invocará a algún depurador de errores instalado en la máquina.

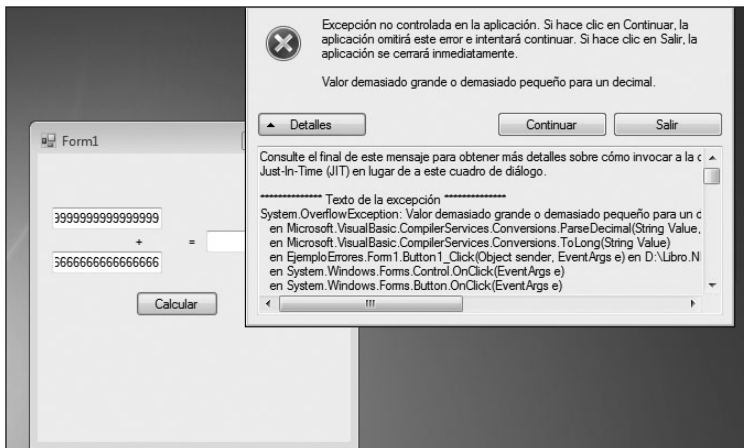


***Figura 4.** Aquí vemos la aplicación compilada y funcionando con un mensaje de excepción no controlada.*

En la **Figura 4** vemos otro error muy común, que no fue prevenido en tiempo de diseño y se presenta cuando el software ya está en uso.

System.Exception

El objeto **Exception** está declarado dentro del framework .NET como tal, siendo su **namespace** completo **System.Exception**. Igualmente, el framework .NET define dos clases genéricas: **System.SystemException** y **System.ApplicationException**. Estas clases no agregan propiedades o métodos sobre la clase base, pero incluyen una manera eficaz de clasificar excepciones dentro de nuestro programa.

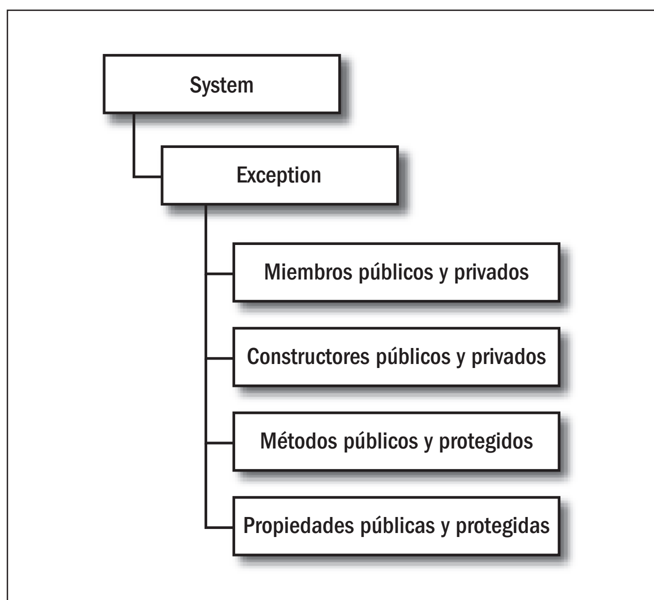


Figura 5. *La clase Exception nos permite conocer en detalle los métodos públicos y privados que luego utilizaremos en el desarrollo de una aplicación.*

Try Catch Finally

En toda la suite Visual Studio tenemos un manejo de errores controlado para sus lenguajes de programación, distinto del antiguo **On Error Goto**. Esta instrucción es conocida como **Try...Catch...Finally**. Veamos a continuación su estructura y analicemos su manera de proceder en la línea de código.



ERRORES DE SINTAXIS

Escribir mal una **palabra clave** o **propiedad** de un objeto provoca una violación en las reglas de Visual Basic 2010 u otros lenguajes de Visual Studio, pero, por suerte, el analizador de código que tiene el IDE señala estos errores mientras el desarrollador escribe los algoritmos, y permite ejecutar la aplicación sólo cuando los hayamos corregido.

```
Sub Procedimiento_O_Funcion()
Try

'Intentando grabar en una Base de datos.
'algoritmo con posibilidad de error.
'Inicio de transacción.
'Ejecución de código SQL.
'Por algún motivo no se pudo ejecutar el SQL.
'Entonces se invoca a Catch.

Catch

'Se produjo una excepción.
'Avisamos al usuario mediante un cuadro de
'mensaje.

Finally

'Como se inició transacción, al no poder
'continuar, debemos deshacer la transacción.

End Try
End Sub
```

En el código ejemplificado tenemos que ejecutar un algoritmo con riesgo de que se produzcan errores, como la inserción o actualización sobre una base de datos. Este paso debe realizarse dentro del bloque **Try**. Al intentar esta operación, puede ocurrir que el servidor de datos no esté disponible o que los registros por modificar ya no existan, con lo cual se produce un error. Este es atrapado por **Catch**, a través del cual debemos informar al usuario que se ha producido un error. **Finally**, en este caso, se ocupa de deshacer la operación anterior para que el usuario tome los recados necesarios antes de volver a llevar a cabo dicha tarea. Iniciemos un nuevo proyecto **Windows Form Application**, con nombre **ErroresEnBBDD**. Para llevar a cabo este ejemplo, haremos uso de la base de datos SQL llamada **Ejemplo.mdf**, que creamos en el **Capítulo 5**. A continuación, en Form1 agregaremos desde el **Cuadro de Herramientas** un control **Button**. Escribimos luego el siguiente código dentro de la aplicación:

```
Imports System.Data.Sql
```

```
Imports System.Data.SqlClient
Public Class Form1
  Dim cnString As String = “Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\databases\ejemplo.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True”

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim con As New SqlConnection(cnString)
    Dim sqlQ As String = “INSERT INTO contactos (Apellido, Nombre,
Empresa, Telefono) VALUES (‘Nicolas Mariano’, ‘Luna’, ‘Nico’s Car Store’,
‘+5411-3527-1453’)”
    Dim cmd As New SqlCommand(sqlQ, con)
    con.Open()
    cmd.ExecuteNonQuery()
    con.Close()
    cmd = Nothing
  End Sub
End Class
```

La finalidad de este código es agregar un registro más a la tabla **contactos**. Presionando **F5**, podremos probar el código escrito.

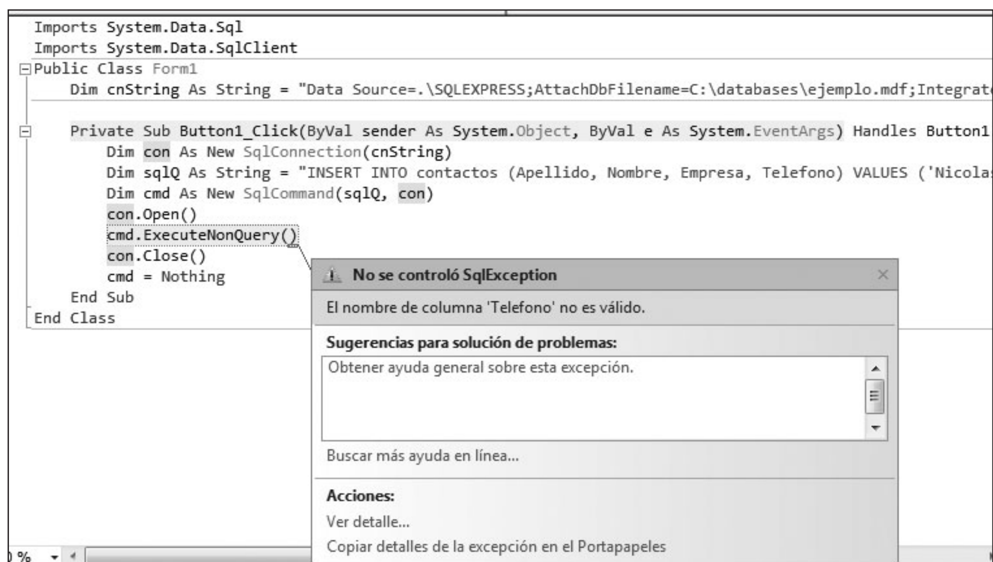


Figura 6. *El código de inserción de un nuevo registro vino acompañado por una excepción no controlada.*

Al escribir la **consulta de inserción** de un nuevo registro, no nos percatamos de que en la tabla **contactos** no teníamos un campo **telefono** para agregar un dato más del contacto. De allí es de donde proviene el error que vemos en la **Figura 6.**

Catch

En el ejemplo llevado a cabo en la **Figura 6**, si el programa hubiese estado compilado, seguramente habría terminado por cerrarse debido al error producido. Para evitar este tipo de situaciones, debemos hacer uso de **Try**, que será la encargada de intentar llevar a cabo la operación, en este caso, de insertar un nuevo registro en la tabla **contactos**; y de **Catch**, que atrapará el error en caso de producirse y notificará al usuario mediante un mensaje que no se ha podido realizar la grabación de un nuevo dato. Corrijamos el ejemplo anterior, agregando el control de errores a nuestro programa. La sintaxis debe quedar como el siguiente código:

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    …
    Try
      con.Open()
      cmd.ExecuteNonQuery()
      con.Close()
      cmd = Nothing
    Catch ex As SqlException
      Dim msg As String = ex.Message
      MessageBox.Show(msg)
    End Try
  End Sub
```

Luego de modificar el código de conexión a la base de datos e intentar la ejecución de la instrucción SQL, probemos que este se lleve a cabo. En realidad, se volverá a producir el error, pero en este caso, estará controlado, y el usuario recibirá un mensaje más claro.



ERR.NUMBER Y ERR.DESCRIPTION

Las instrucciones antiguas del objeto **Err** se siguen manteniendo. En los casos de uso de **Try…Catch When…** para el manejo de excepciones, con **Err.Number** podemos llegar a controlar mejor el flujo de la excepción, y de esta manera, dar un mensaje y una instrucción más claros al usuario, para que continúe usando la aplicación.

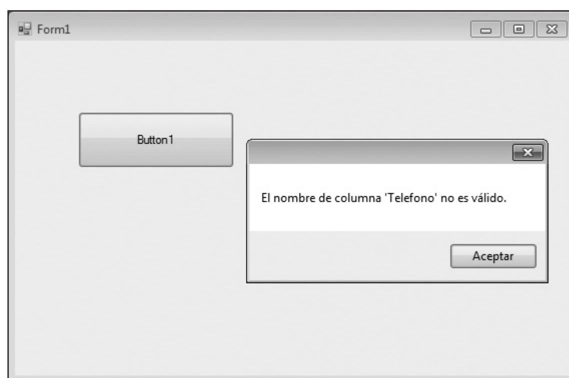


Figura 7. *El mismo error provocado en la* ***Figura 6****, pero esta vez controlado por un mensaje más claro.*

Hemos logrado llevar los mensajes de error dentro del control de nuestra aplicación. Estos se verán algo más claros cuando el usuario los reciba, y con algunos ajustes más, podremos hacer que sea él quien decida sobre el flujo que debe tomar la aplicación.

Al momento de usar **Catch**, declaramos una variable **ex as sqlException**. Visual Basic contiene varios tipos de datos **Exception**. En este caso específico, al estar trabajando sobre una base de datos, dentro del algoritmo, cualquier error que ocurriese tendría que ver con un error de base de datos. Por eso utilizamos una declaración **ex**, que sea del tipo **sqlException**, ya que agrupa todo error que provenga de **un** manejo de datos a través de sentencias **SQL**, en este caso exclusivamente de SQL Server. Dentro de **sqlException** estarán todos los errores comunes a ellas. La clase **sqlException** pertenece al **namespace System.Data.sqlClient**. Para hacer un manejo prolijo de **sqlException** cuando trabajamos con SQL Server, podemos consultar la tabla **sysmessages** alojada en la base de datos **Master**. Esta tabla contiene todos los errores que pueden ocurrir con SQL Server. Si llamamos a esta tabla mediante una instrucción **SELECT**, nos encontraremos con unos 80.000 registros. Tengamos en cuenta que guarda una colección de **sqlException** para todos los idiomas en los que fue publicado el sistema operativo **Windows**. Para ver solo el español, debemos realizar un filtro del campo **mslangid = 3082**.



DESHABILITAR PUNTOS DE INTERRUPCIÓN

El nombre real de los puntos de interrupción es **glifos**, y se ubican en el margen izquierdo de la ventana de código. Se pueden deshabilitar haciendo un clic con el botón derecho del mouse y seleccionando **Deshabilitar punto de interrupción**. Desde el menú **Depurar**, tenemos la opción **Habilitar/Deshabilitar todos los puntos de interrupción**.

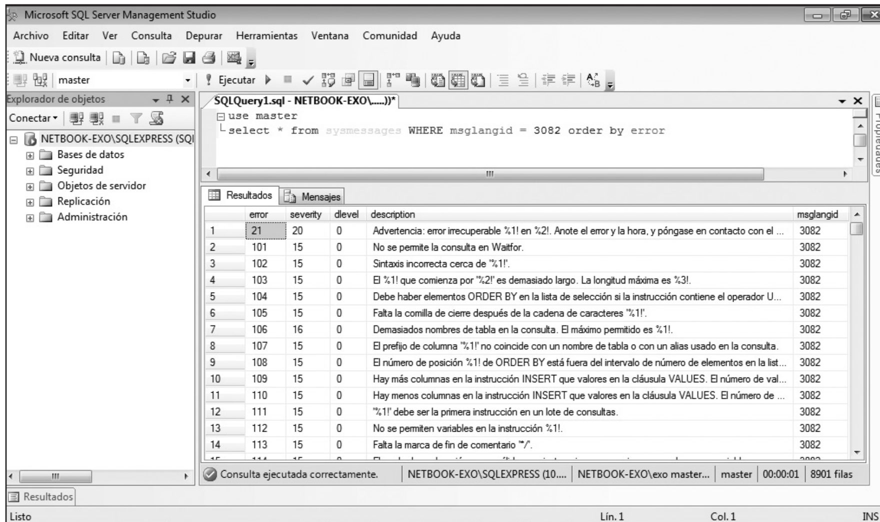


***Figura 8. SQL Server 2008** agrupa en la tabla **sysmessages**,*
*todos los errores posibles que ocurren a través de sentencias **sql**.*

When como palabra clave dentro de Catch

La palabra clave **When**, dentro de la instrucción **Catch**, permite llevar un control más exacto para poder darle un mensaje claro al usuario de la aplicación, y sobre la base del tipo de error, permitir que él realice una instrucción. Nosotros, como programadores, debemos aprovechar esta palabra clave para saber cómo continuar el flujo de nuestro algoritmo en caso de que se produzca un error. Modifiquemos el ejemplo, cambiando el contenido de la instrucción SQL, corrigiendo el campo que no existe, pero provocando otro error dentro de la instrucción de insertar un nuevo registro. Escribamos lo siguiente en la declaración de la instrucción **sqlQ**:

```
Dim sqlQ As String = "INSERT INTO contactos (Apellido, Nombre, Empresa)
VALUES ('Nicolas Mariano', 'Luna', 'Nicolas Mariano Luna Car Store -
Repuestos y tunning de autos')"
```



DEBUG.PRINT EN VENTANA INMEDIATO

Si deseamos ver cómo cambia el valor de una variable, podemos invocar en modo de depuración a la sentencia **Debug.Print nombre_de_variable**. Cada vez que el programa cambie el valor de dicha variable, esto creará una línea en la ventana **Inmediato**, mostrando su nuevo valor. Esta manera se hereda desde las primeras versiones de Visual Basic.

La instrucción siguiente, aunque a simple vista no lo parece, provocará otro error en el código. Guardamos nuestro proyecto, y, a continuación, ejecutamos el mismo presionando la tecla **F5** para probar qué sucede.

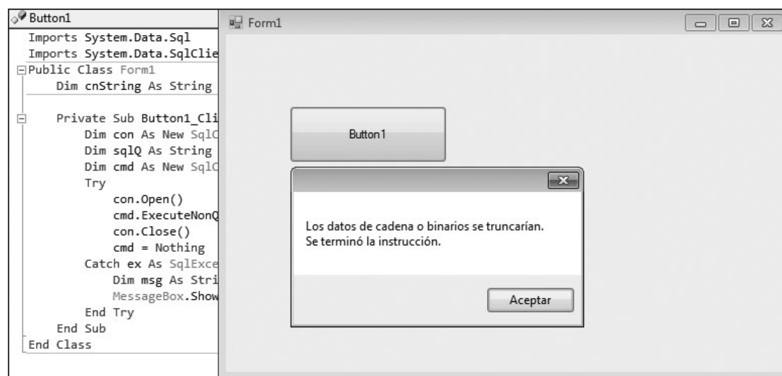


Figura 9. *Se ha provocado un error, dado que no se tuvo en cuenta el máximo de caracteres permitidos en el campo* ***Empresa****.*

El tipo de error visto en la **Figura 9** podrá controlarse gracias a la instrucción **When**. Vamos a cambiar el código de **Catch** a través del cual verificamos si se produce este error, para darle un mensaje más preciso al usuario que el listado por el sistema:

```
Catch When Err.Number = 5 'Excedido de caracteres
   Dim msg As String = "Se produjo un error debido a que uno de los datos
ingresados, excede el tamaño máximo de caracteres permitidos dentro de
algún campo. Por favor, reduzca la cantidad de caracteres e intente grabar
nuevamente."
   MessageBox.Show(msg, "Error grabando")
End Try
```

Seguramente el lector se preguntará si es posible anidar más de un **Catch When**, dado que, en determinadas situaciones, puede provocarse más de un mismo tipo de error. Efectivamente, es posible hacerlo. Cambiemos el ejemplo, iniciando un nuevo proyecto del tipo **Windows Forms Application**. Agreguemos al **Form1** un control **pictureBox** y un control **Button**. Dentro del evento **button1_click()** vamos a añadir el siguiente código:

```
    Dim archivo As String = "D:\julyynico.jpg"
    Try
      PictureBox1.Image = System.Drawing.Bitmap.FromFile(archivo)
```

```
     Catch When Err.Number = 53
        Dim msg As String = "El archivo no existe o no se pudo encontrar la
ruta de acceso."
        MessageBox.Show(msg)
     Catch When Err.Number = 7
        Dim msg As String = "Error de memoria, o el archivo especificado no
es una imagen."
        MessageBox.Show(msg)
     End Try
```

Veamos qué hace este código. En primer lugar, declaramos una variable de tipo **String**, a la cual le asignamos un valor inicial de una ruta hacia un archivo de imagen. Luego iniciamos la instrucción **Try** e intentamos cargar la imagen especificada en la variable **archivo**. Para ejemplificar este error, se utilizó un pendrive conectado a la computadora, que tiene como **Letra de unidad** la **D:**. Si no tenemos un **pendrive** o dicha unidad en una lectora de **CD/DVD**, podemos redireccionar el ejemplo a la unidad **C:**, que suele ser el disco rígido de la computadora. En principio, la imagen **d:\julyynico.jpg** no existe. Este error se atrapa con **Err.Number = 53**, para indicar al usuario que no existe la ruta o la imagen especificada. Ahora, para probar en qué momento se invoca **Err.Number = 7**, copiamos a dicha unidad un archivo, que puede ser un documento **.DOC** o **.XLS**, o incluso un **.PDF**, y cambiamos su nombre y extensión por los de la imagen utilizada. A continuación, intentamos cargar la imagen otra vez.



***Figura 10.** Aquí nos encontramos con el error número 7, cuyo dato es que el archivo sí existe, pero no es una imagen válida.*

Finally

Pocos hacen uso de esta cláusula, pero no está de más implementarla en el control de errores. **Finally** puede utilizarse claramente para volver a vaciar variables del tipo

Global o restablecer a **Nothing** determinados objetos para que no ocupen espacio en memoria, sobre todo, cuando trabajamos con archivos o conexiones de bases de datos. Veamos a continuación el siguiente código donde se utiliza **Finally**:

```
    Try
      PictureBox1.Image = System.Drawing.Bitmap.FromFile(archivo)
    Catch When Err.Number = 53
      Dim msg As String = "El archivo no existe o no se pudo encontrar la
ruta de acceso."
      MessageBox.Show(msg)
    Catch When Err.Number = 7
      Dim msg As String = "Error de memoria, o el archivo especificado no
es una imagen."
      MessageBox.Show(msg)
    Finally
      archivo = ""
      PictureBox1.Image = Nothing
    End Try
```

Por más que el **Try** se ejecute correctamente o no, al momento de llegar a **Finally**, el código contenido dentro de esta instrucción se ejecutará igual. Con esto, se vaciará el contenido de **PictureBox1** y la variable **archivo** quedará sin valor asignado.
En otras oportunidades, seguramente al ejecutar la instrucción **Catch**, debamos preguntarle al usuario si desea reintentar o no, mediante la instrucción **MessageBox.Show()** acompañando el parámetro de botones **MessageBoxButtons.YesNo**. Para el caso de reintentar, deberemos ejecutar otra vez un código, y para el caso de no reintentar, deberemos detener el resto del algoritmo, aunque este puede contener código adicional que pueda perjudicar el funcionamiento del programa, dentro de la instrucción **Finally**. Para evitar seguir procesando este control de error, podemos hacer uso de **End Try**. Esta instrucción, al igual que **End Sub** y **End Function**, saldrá del bloque **Try**, salteando incluso **Catch** y **Finally**.



ANIDAR CONTROL DE EXCEPCIONES

Una instrucción **Try** se lleva a cabo, y según el éxito o no que tenga, se llama a **Catch**. En algunos casos, tal vez sobre la base de un error específico debamos cambiar la dirección del algoritmo, con lo cual tendremos que utilizar un nuevo **Try...Catch** anidado dentro del primero. Esto puede hacerse como si fuesen funciones o procedimientos recursivos.

Instrucción Throw

La cantidad de excepciones que pueden aparecer dentro de una aplicación de Visual Basic no está limitada a las controladas por el motor del framework .NET. También podemos crear nuestras propias excepciones y notificar sobre ellas a los usuarios. La instrucción **Throw** sirve, justamente, para eso. Veamos cómo implementarla a través de un código de ejemplo. Creamos un nuevo proyecto del tipo **Windows Form Application**, agregamos una caja de texto, un botón, y dentro del control **Button** escribimos el siguiente código:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
     If Not IsNumeric(TextBox1.Text) Then Throw New Exception("Este mensaje
de error lo invento yo, Fernando Luna.")
     Dim i As Integer = 100
     Dim j As Long = Trim(TextBox1.Text)
     MessageBox.Show(i + j)
End Sub
```

La instrucción **Throw** nos permite crear excepciones personalizadas sobre la base de nuestros propios controles. Este código nos muestra cómo crear una excepción para el caso de que hubiésemos ingresado una cadena alfanumérica en un **textBox** cuando se esperaba un valor numérico.

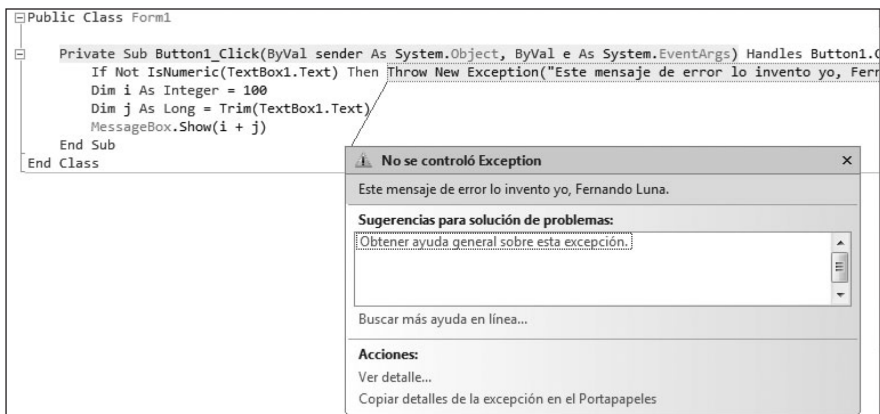


***Figura 11.** Dependiendo de los errores personalizados, podremos controlar la aplicación mediante las excepciones basadas en ellos.*

El objeto **Err** se mantuvo con el correr de las versiones de Visual Basic, dado que nos permite obtener información del control de errores cada vez que ocurre un error en tiempo de ejecución de nuestra aplicación. Las propiedades más utilizadas

de la clase **Error** son **Number** y **Description**. La primera entrega el número de error, y la segunda, una descripción de él. Con el método **Err.Clear** podemos restablecer la clase Err como si nunca hubiese ocurrido un error, aunque esto no es necesario si utilizamos el **Err** dentro de un bloque **Catch**, dado que los bloques **Catch** solo se accionan cuando ocurrió el error en tiempo de ejecución dentro de un bloque **Try** cercano a él. Cada objeto, control y bibliotecas que utilizamos dentro de Visual Basic proporciona sus propios números y descripción de los errores. De todos modos, Visual Basic controla los errores típicos que pueden ocurrir comúnmente. Se reserva los números de error desde el 1 al 1000 para manejar de manera interna los errores típicos. Veamos en la **Tabla 1** los errores más comunes que se conocen al programar sistemas:

NÚMERO DE ERROR	DESCRIPCIÓN PREDETERMINADA
5	El argumento o invocación de un procedimiento no es válido
6	Desbordamiento
7	Memoria insuficiente
9	El subíndice está fuera del intervalo
11	División entre cero
13	No coinciden los tipos
48	Error al cargar la biblioteca DLL
51	Error interno
52	Nombre o número de archivo incorrecto
53	Archivo no encontrado
55	El archivo está abierto
57	Error de Entrada-Salida del dispositivo
58	El archivo ya existe
61	Disco lleno
62	Se sobrepasó el final del archivo
67	Demasiados archivos
68	Dispositivo no disponible
70	Permiso denegado
71	Disco no preparado
74	No se puede cambiar el nombre con una unidad de disco diferente
75	Error de acceso a la ruta o al archivo
76	No se encuentra la ruta de acceso
91	Variable de objeto o de bloque With no establecida
321	Formato de archivo inválido
322	No se puede crear el archivo temporal necesario
380	Valor de propiedad inválido
381	El índice de la matriz de propiedades no es válido
422	No se encuentra la propiedad

NÚMERO DE ERROR	DESCRIPCIÓN PREDETERMINADA
423	No se encuentran la propiedad ni el método
424	Se requiere un objeto
429	No se puede crear el componente ActiveX
430	Esta clase no admite automatización o no admite la interfaz esperada
438	El objeto no admite esta propiedad o método
440	Error de automatización
460	El formato de portapapeles es inválido
461	No se encuentra el método o miembro de datos
462	El equipo del servidor remoto no existe o no está disponible
463	La clase no está registrada en el equipo local
481	La imagen no es válida
482	Error de impresora

Tabla 1. *Con esta lista básica es posible controlar la mayoría de los errores típicos al escribir código en Visual Basic.*

Cómo forzar una excepción

Generalmente, en determinados procesos de evaluación de código o de depuración, nos encontramos con la necesidad de producir nuestros propios errores, requeridos en determinados casos para llevar el flujo del algoritmo hacia la instrucción **Try…Catch**. Para hacerlo, debemos utilizar la técnica llamada **arrojar** o **provocar excepciones**, que implica emplear el método **Err.Raise** junto con algún número de error. Suponiendo que queremos copiar un archivo desde el disco **C:** hacia el **D:**, debemos crear el algoritmo correspondiente, y al momento de realizar la copia en sí, esta debe ir dentro de un evento **Try…Catch**. Mirando la tabla anterior, podemos verificar que muchos de estos errores apuntan al manejo de archivos o directorios. Suponiendo que realizamos la copia de un archivo desde una unidad hacia la otra, y queremos provocar un error de **Disco lleno**, debemos realizar lo siguiente:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim archivo As String = “D:\julyynico.jpg”
    Try
      System.IO.File.Copy(archivo, “C:\julyynico.jpg”)
      Err.Raise(61)
    Catch ex As Exception
      MessageBox.Show(ex.Message)
    End Try
End Sub
```

Tengamos en cuenta que si llamamos a **Err.Raise** después de ejecutar el evento **Copy**, y durante la ejecución de este no se provoca ningún error, el mensaje de **Disco lleno** será mostrado, pero el archivo será copiado igual al destino especificado. Siempre busquemos llamar a los errores que conocemos, teniendo en cuenta que la instrucción anterior puede no provocar errores y ejecutarse sin problema alguno.



***Figura 12.** Un error provocado a propósito por el método Err.Raise. Tengamos en cuenta que el código anterior puede ejecutarse sin problemas aunque hayamos llamado al error.*

DEPURACIÓN DE APLICACIONES

Hasta ahora descubrimos cómo manipular excepciones ocurridas en el funcionamiento de una aplicación. El mundo del desarrollo es amplio y permite a través de herramientas manipular un código limpio claro y lo menos complejo posible, para que nuestras aplicaciones se ejecuten óptimamente. Muchos manuales de programación muestran depurar aplicaciones al final de la obra, pero realmente escribir un buen código se logra depurando exhaustivamente el código, así el mismo llega con la menor cantidad de errores hacia la versión beta. Ser buen programador no se logra solo con conocimientos de programación, sino también de depuración. Muchas empresas poseen equipos de testing encargados de exprimir nuestros desarrollos, pero cuando trabajamos solos durante años, difícilmente dispongamos de gente que testee nuestros productos. Veamos, a continuación, las herramientas que Visual Basic 2010 ofrece para la depuración de errores.

Herramientas de depuración

Visual Basic 2010 cuenta con una serie de herramientas de depuración y revisión de aplicaciones, que pueden ponerse en práctica mientras escribimos nuestro software o cuando lo ejecutamos. Veamos a través de la **Figura 13** cuáles son.

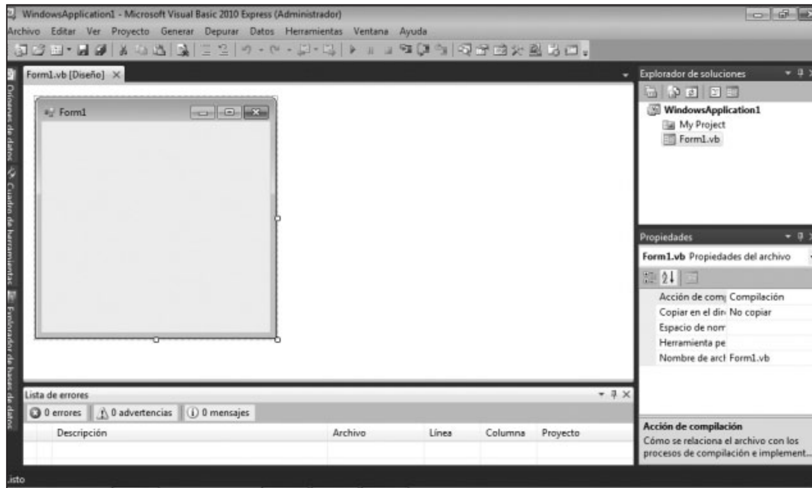


Figura 13. *Las principales herramientas de depuración tienen su propia barra, para que sea más cómodo usarlas.*

- **Iniciar depuración**: permite arrancar la ejecución de una aplicación Microsoft Visual Basic en modo de depuración. Cualquier error que ocurra será interrumpido en ese instante. En los capítulos anteriores, muchas veces hemos ejecutado nuestras aplicaciones presionando la tecla **F5**. Esto mismo equivale a iniciar la depuración de la aplicación.
- **Pausar depuración:** permite establecer una pausa definida por el usuario en cualquier momento de la ejecución del programa en modo depuración. Pausar la depuración de nuestra aplicación no implica perder sus valores ya establecidos. Cada variable que esté declarada en un **Form** o en modo **Global**, y que tenga asignado un valor, los mantendrá hasta que sea destruida o el programa sea detenido. Pausar una depuración no hará que se pierda ningún valor.
- **Parar depuración**: equivale a detener la ejecución del programa.
- **Paso a paso por instrucciones**: permite ejecutar la aplicación siguiendo paso a paso cada línea de código que escribimos. Esto equivale a presionar la tecla **F8**. Esta opción paso a paso permitirá que se recorra cada línea de código, y no se ejecutará la siguiente hasta tanto volvamos a presionar este botón de la barra de herramientas, o en su defecto, **F8** otra vez.
- **Paso a paso por procedimientos**: funciona igual que la opción anterior, con la diferencia de que procesa todo un bloque de código junto, y no línea por línea. Funciona también presionando las teclas **SHIFT + F8**.
- **Paso a paso para salir**: el programa debe estar en modo **Pausa** o **Break** para que podamos hacer uso de esta opción.
- **Examinador de objetos**: permite verificar todos los componentes (propiedades, métodos, eventos) de cada objeto, clase y namespaces que integran Visual Basic 2010. Se invoca también presionando la tecla **F2**. Si, por ejemplo, buscamos la

palabra **Format** dentro del **Examinador de Objetos**, veremos un detalle de todas las implementaciones de **Format** que utiliza la suite Visual Studio.

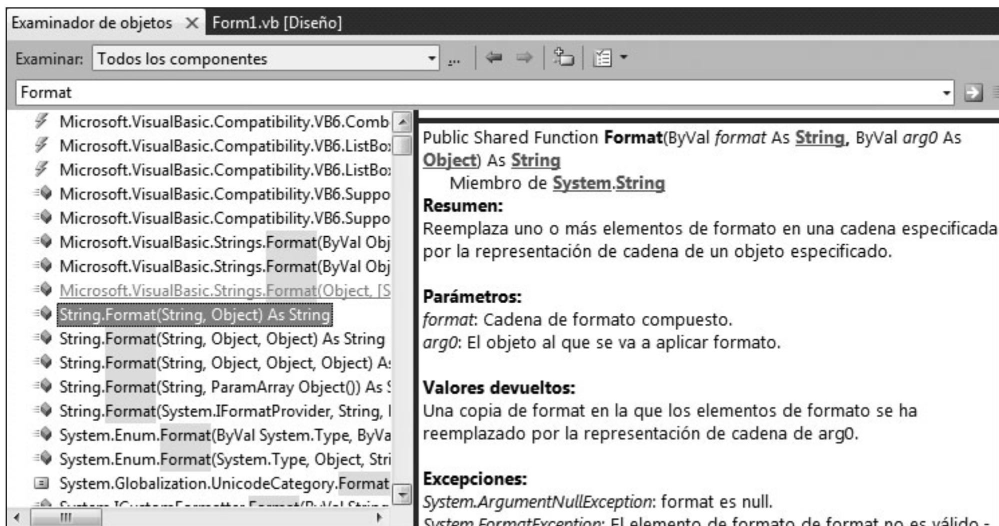


***Figura 14.** Revisión completa de la palabra Format, gracias al **Examinador de Objetos**.*

- **Lista de errores**: es una colección de cada error que ocurre durante la escritura o ejecución de la aplicación. Esto solo se activa cuando hay errores no corregidos en el código fuente. Veamos a continuación algunos ejemplos. Creamos un nuevo proyecto del tipo **Windows Forms Application**. Ingresamos en **Form1** y escribimos el siguiente código:

```
Public Class Form1

  Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim archivo As String
    Dim ejemplo As Boolean
    Dim numError As Int16
    Dim con As sqlConnection
    con.open()
  End Sub
End Class
```

Si bien en este código tendríamos que ver algunas advertencias no graves, el principal error es que estamos declarando una conexión Sql y abriéndola sin siquiera

haber importado el namespace **System.Data** y **System.Data.SqlClient**. Veamos en la **Figura 15** cómo se detalla el error.

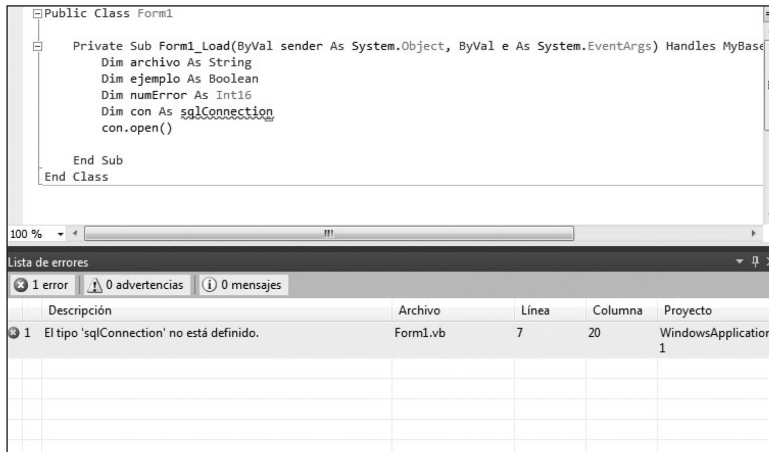


***Figura 15.** Lista de errores. La ventana común nos mostrará dónde nos equivocamos la mayoría de las veces.*

- **Ventana de inmediato**: esta opción nos permite verificar el valor de variables, pausar la aplicación, llamar a procedimientos o funciones, establecer el valor de variables u objetos, y otras opciones más durante la depuración de una aplicación, siempre y cuando esta se encuentre en modo pausado. Para ver en ejecución la ventana de inmediato, ejecutamos un proyecto ya desarrollado, como Drugstore. Luego abrimos la pantalla de **Ventas**. Dentro de la clase **frmVenta** declaramos una variable de tipo **String**, llamada **cnString**. Si deseáramos ver el valor de dicha variable, primero tendríamos que presionar **Pausa**, y **F8** sobre la aplicación, luego invocar la apertura de **frmVenta**, pulsando el botón **Ventas**. Por último, habría que ejecutar parte del código hasta que el valor **cnString** se hubiera creado y asignado, sin llegar al final del proceso de **form_load()**. Para ver el valor de **cnString**, debemos invocar a la ventana inmediato la cual, de no estar visible, a través de las teclas **CTRL + G**. Luego, escribimos el siguiente código en la ventana inmediato:

```
?cnString
```

Como resultado, obtenemos:

```
"Data Source=.\SQLEXPRESS;AttachDbFilename=C:\databases\drugstore.mdf;Inte
grated Security=True;Connect Timeout=30;User Instance=True"
```

El valor de **cnString** puede visualizarse mediante la **Ventana de Inspección**. Por supuesto, también podemos cambiar cualquier valor desde allí mismo. Probemos a cambiar el nombre de la base de datos por uno inexistente, para ver cuándo se provoca el error de conexión. Pongamos el código listado a continuación en en otro renglón de la ventana de inmediato:

```
cnString =  "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\databases\drugs
tore_2011.mdf;Integrated Security=True;Connect Timeout=30;User
Instance=True"
```

Luego de escribir esto, le damos un **Enter**. A continuación, ejecutamos el código con la tecla **F5**. Iniciamos una nueva venta, y grabamos a continuación la misma. Nos saldrá un error diciendo que no existe la base de datos especificada y que no se pudo conectar con ella. Esto ocurre porque, en tiempo de ejecución de la aplicación, hemos cambiado el valor real de la variable que contenía la cadena de conexión.

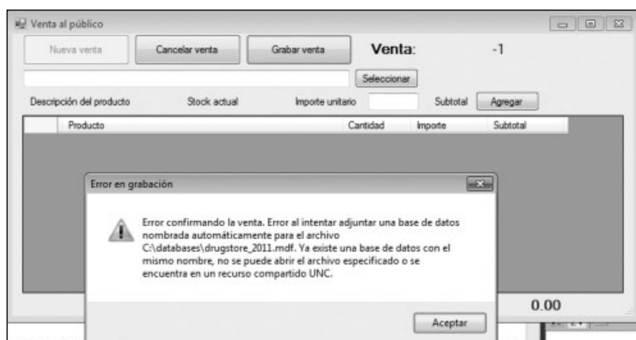


Figura 16. *Debido al cambio de valor sobe la variable de conexión, se desemboca finalmente en un mensaje de error.*

Puntos de interrupción

Los puntos de interrupción son útiles para detener la ejecución del software en determinado punto, y así poder continuarlo paso a paso, o verificar el valor de cierto objeto o variable antes y después de que la línea de ejecución pasa por él. Para crear un punto de interrupción, primero debemos establecer en qué parte del código vamos a crearlo. Abrimos otra vez el proyecto **Drugstore**. Sobre el **Explorador de soluciones** buscamos el formulario **frmVenta**. Hacemos clic sobre él y presionamos la tecla **F7**. Esto hará que se abra la ventana de código fuente, mostrando toda la clase **frmVenta** y el código ingresado en cada uno de sus componentes. Buscamos a continuación la función **graboNuevoNro()** y, en la línea **graboNuevoNro = False**, posicionamos el cursor del mouse y presionamos la tecla **F9**. Aparecerá un punto rojo oscuro

sobre el lateral izquierdo de la ventana de código y un sombreado sobre esta línea de código del mismo color, tal como lo muestra la **Figura 17**.

```
Imports System.Data.Sql
Imports System.Data.SqlClient

Public Class frmVenta
    Dim cnString As String = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\databases\drugstore.mdf;Ir

    Function graboNuevoNro() As Boolean
        Dim con As New SqlConnection(cnString)
        Dim sqlQ As String = "UPDATE numeracion SET numticket = (numticket + 1)"
        Dim cmd As New SqlCommand(sqlQ, con)
        graboNuevoNro = False
        Try
            con.Open()
            cmd.ExecuteNonQuery()
            graboNuevoNro = True
        Catch ex As Exception
            MessageBox.Show("Error confirmando la venta. " & ex.Message, "Error en grabación", Messag
            graboNuevoNro = False
        End Try
    End Function

    Sub cambioBotones()
        btnNuevaVenta.Enabled = Not btnNuevaVenta.Enabled
        btnCancelar.Enabled = Not btnCancelar.Enabled
        btnGrabar.Enabled = Not btnGrabar.Enabled
```

***Figura 17.** De esta manera queda establecido el punto de interrupción en nuestra aplicación.*

En modo de diseño, esto no cumple ninguna función. Debemos ejecutar la aplicación presionando **F5** para probar esta funcionalidad. Ejecutamos la aplicación y abrimos el módulo de **Ventas**. A continuación, presionamos el botón **Nueva venta** y agregamos algunos productos para llevar a cabo una nueva venta. Al presionar el botón **Grabar venta**, veremos que la ejecución del programa se detiene precisamente donde establecimos el punto de interrupción. Si posicionamos el cursor sobre alguna variable declarada, podremos apreciar el valor correspondiente a través de una etiqueta **ToolTip**. Esto sirve tanto para ver el valor de la variable donde establecimos la detención de ejecución del código, como para ver el valor de otra variable que ya se haya ejecutado.

```
Imports System.Data.Sql
Imports System.Data.SqlClient

Public Class frmVenta
    Dim cnString As String = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\databases\drugstore

    Function graboNuevoNro() As Boolean
        Dim con As New SqlConnection(cnString)
        Dim sqlQ As String = "UPDATE numeracion SET numticket = (numticket + 1)"
        Dim cmd As New SqlCommand(sqlQ, con)
        graboNuevoNro = False
        Try          graboNuevoNro | False
            con.Open()
            cmd.ExecuteNonQuery()
            graboNuevoNro = True
        Catch ex As Exception
            MessageBox.Show("Error confirmando la venta. " & ex.Message, "Error en grabación",
            graboNuevoNro = False
        End Try
    End Function

    Sub cambioBotones()
        btnNuevaVenta.Enabled = Not btnNuevaVenta.Enabled
        btnCancelar.Enabled = Not btnCancelar.Enabled
```

***Figura 18.** La Etiqueta **ToolTip** nos permite conocer información acerca del estado y valor de las variables utilizadas.*

Si revisamos el contenido de una variable aún no ejecutada, como **ex.Message**, que está dentro del bloque **Catch**, veremos que este objeto todavía no tiene asignado ningún valor. De lo contrario, si revisamos el valor de la variable **sqlQ**, veremos que en ella se encuentra cargada la sentencia **Sql** que sumará 1 al valor de la tabla **numeracion**.

```
Imports System.Data.Sql
Imports System.Data.SqlClient

Public Class frmVenta
    Dim cnString As String = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\databases\drugstore.mdf;Integrat

    Function graboNuevoNro() As Boolean
        Dim con As New SqlConnection(cnString)
        Dim sqlQ As String = "UPDATE numeracion SET numticket = (numticket + 1)"
        Dim cmd  sqlQ  "UPDATE numeracion SET numticket = (numticket + 1)"
        graboNuevoNro = False
        Try
            con.Open()
            cmd.ExecuteNonQuery()
            graboNuevoNro = True
        Catch ex As Exception
            MessageBox.Show("Error confirmando la venta. " & ex.Message, "Error en grabación", MessageBoxBu
            graboNuevoNro = False
        End Try
    End Function

    Sub cambioBotones()
        btnNuevaVenta.Enabled = Not btnNuevaVenta.Enabled
```

Figura 19. *La Etiqueta **ToolTip** muestra el valor de la variable **sqlQ** dentro del código de nuestra aplicación.*

Podemos establecer todos los puntos de interrupción que necesitemos dentro del código de la aplicación, y dejarlos marcados. Estos se mantendrán siempre y cuando no cerremos el proyecto. Si lo cerramos, se perderá toda referencia a ellos. Cuando debemos analizar muchos estados de variables y objetos, siempre es conveniente marcar todos los puntos de interrupción que nos parezcan. Podremos ejecutar dicho programa presionando **F5**, luego de evaluar uno de los puntos de interrupción.

```
        Reader.Close()
    End Sub

    Private Sub btnGrabar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
        If graboNuevoNro() Then
            cambioBotones()
            LimpioCampos()
            Dim msg As String = "Se ha grabado correctamente la venta."
            MessageBox.Show(msg, "Nueva venta", MessageBoxButtons.OK, MessageBoxIcon.Information)
        End If
    End Sub
    Sub LimpioCampos()
        Me.VwventasBindingSource.Filter = "nroventa = 0"
        lblVenta.Text = "0"
        txtSelecciono.Text = ""
        lblTotal.Text = "0.00"
        lblDesc.Text = "Descripción del producto"
        lblStk.Text = "Stock actual"
        lblImporte.Text = "Importe unitario"
        txtCantidad.Text = ""
        lblSubt.Text = "Subtotal"
    End Sub
```

Figura 20. *En el evento **Grabar_Click()** podemos ver distintos puntos de interrupción, incluyendo uno que está dentro del procedimiento **LimpioCampos()**.*

Pila de llamadas

La pila de llamadas nos permite ver el llamado a las funciones o procedimientos que actualmente están dentro de la pila. Se presenta como una ventana o solapa dentro del

IDE de Visual Basic 2010. Para activarla, debemos tener el programa en modo interrupción, o pausado, y desde el menú **Depurar/Ventanas/Pila de llamadas** activar dicha solapa. Dentro de la información que nos brinda, veremos el nombre de la función o procedimiento, junto con información adicional, como nombre del módulo, número de línea y nombres, tipos y valores de parámetros que estén dentro de ellos. Una flecha de color amarillo indica el marco de pila donde se ubica actualmente el puntero en ejecución. Si no hubiese símbolos de depuración disponibles para una parte de la pila de llamadas, dicha ventana mostraría un mensaje similar al siguiente:

```
[Es posible que los siguientes marcos no estén o sean incorrectos. No se
han cargado símbolos para nombredelibrería.dll]
```

Veamos a continuación la pila de llamadas en la aplicación **Drugstore** pausada justo en el módulo de grabación.

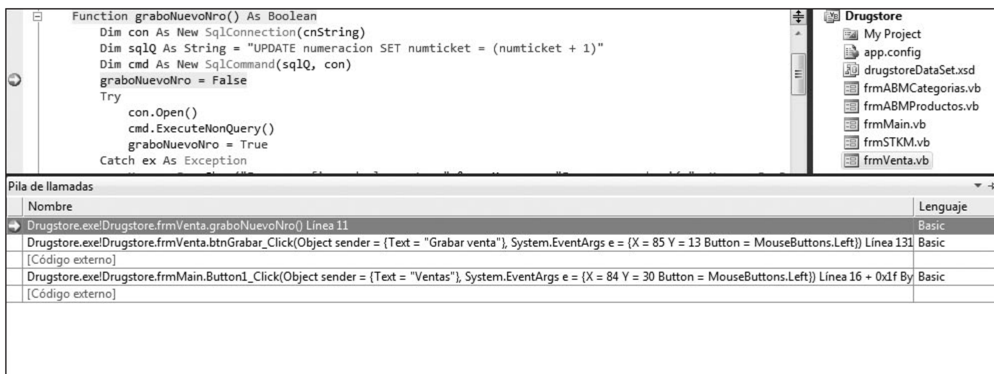


***Figura 21.** Haciendo doble clic sobre algún renglón de la ventana **Pila de llamadas**, podemos posicionar el cursor directamente sobre esta acción.*

Debug y Trace

Mientras depuramos aplicaciones, toda la información que se genera durante la ejecución de los algoritmos, su traza y depuración se envía a una ventana de salida



CAMBIO DE MARCO DE PILA

Si deseamos cambiar a otro marco de pila, desde la solapa **Pila de llamadas** y sobre el objeto de código y datos que deseamos ver, hacemos un clic con el botón derecho del mouse y seleccionamos **Cambiar a marco**. Aparecerá una flecha verde, y el puntero se mantendrá en el marco original, marcado con su respectiva flecha amarilla.

de Visual Basic 2010. Si deseamos incluir características de traza, debemos compilar nuestros proyectos utilizando la directiva **Trace**, que debe habilitarse desde el compilador. Esto hará que el código de traza se compile sobre la versión **RELEASE** de nuestro proyecto. Los métodos de traza y depuración disponen de atributos asociados. Si tenemos un atributo condicional para traza, éste toma el valor **True**, y todas las instrucciones de la traza serán incluidas dentro del ensamblado, sea **.EXE** o **.DLL**. Si dicho atributo es **False**, las instrucciones no serán incluidas. Si queremos cambiar las opciones de compilación de nuestros proyectos, debemos hacerlo desde el **Cuadro de diálogo** de la **Página de propiedades**. Para eso, sobre el **Explorador de soluciones** hacemos clic con el botón derecho del mouse, y desde el menú contextual, seleccionamos **Propiedades**. Vamos a la pestaña **Compilar** y presionamos el botón ubicado al pie de la página, denominado **Opciones de compilación avanzadas**. Se abrirá una ventana titulada **Configuración de compilación avanzada**. Luego, nos queda marcar y desmarcar las opciones que nos interesan. Aceptamos, y listo. Ya estará habilitado el modo **Debug and Trace** en nuestro proyecto.

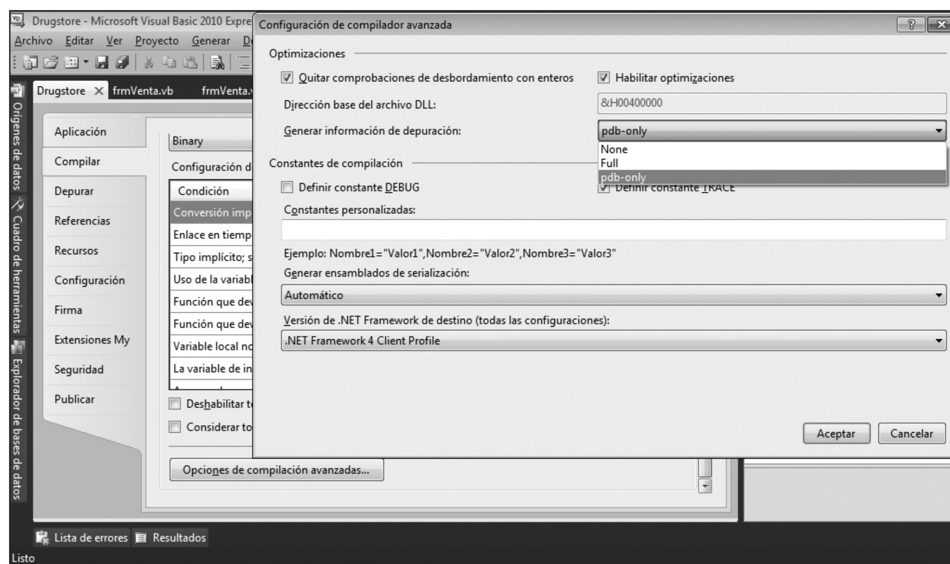


***Figura 22.** Habilitar opciones de Debug and Trace en nuestros proyectos.*

Modificadores de seguimiento

Utilizando la estrategia de **Debug and Trace**, siempre debemos pensar en la manera en que se presentarán los resultados. Para hacerlo, podemos aprovechar los modificadores de seguimiento, que son métodos usados para marcar la traza, escribiendo información en los agentes de escucha. Dichos métodos son: **Write**, **WriteIf**, **WriteLine**, **WriteLineIf**, **Assert** y **Fail**. Se dividen en categorías, **Write**, **WriterLine** y **Fail**, que envían resultados de manera incondicional; mientras que **WriteIf**, **WriteLineIf** y **Assert** comprueban condiciones booleanas, y de esta manera escriben los valores de

su condición siempre y cuando tengan un .valor **True**. **Assert** es la única que escribirá un resultado si la condición toma un valor **False**.

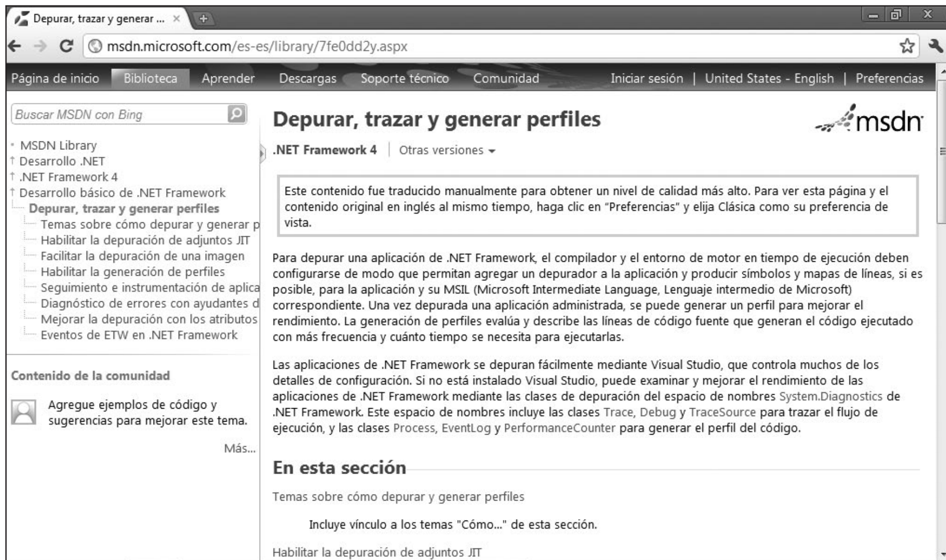


***Figura 23.** El portal MSDN contiene información detallada de Debug and Trace para cada versión específica de Visual Studio .NET.*
http://msdn.microsoft.com/es-es/library/7fe0dd2y.aspx

... RESUMEN

Durante este capítulo pudimos conocer de qué manera aprovechar la capacidad de controlar errores en nuestra aplicación, qué tipos de clases utilizar y en qué momento implementar cada una. También repasamos los puntos de interrupción y las herramientas de depuración que vienen incluidas tanto en Visual Basic 2010 como en el resto de la suite Visual Studio. Las herramientas de depuración nos permiten controlar nuestra aplicación en la fase de desarrollo y corroborar que la asignación de variables, o la ejecución de cierto código, se lleve a cabo tal como lo razonamos al momento de escribirlo.



ACTIVIDADES

EJERCICIOS PRÁCTICOS

1. Utilizando el proyecto **Drugstore**, en modo ejecución, investigue qué ocurre cuando se detiene en un punto de interrupción y presiona **CTRL + F8**.

2. Investigue si es posible o no seguir utilizando la sentencia **On Error Goto**, usada en las versiones previas a .NET.

3. Cree una aplicación con dos o más sentencias distintas, y en una de ellas fuerce un error y utilice **Exit Sub** para salir del procedimiento. Luego evalúe qué ocurre al seguir utilizando la aplicación.

4. Pruebe qué ocurre poniendo **Err = 0** dentro de un evento **Try Catch**.

5. Investigue, siguiendo la línea del ejemplo anterior, qué ocurre si presiona **CTRL + F9**.

6. Investigue utilizando la red Internet qué es un glifo y para qué se utiliza.

7. Establezca varios puntos de interrupción dentro de todo el proyecto y vea de qué manera se pueden quitar todos juntos en vez de tener que hacerlo de a uno.

8. Investigue el uso de la sentencia **Debug.Print** dentro del código del proyecto y verifique cómo trabaja esta sentencia en el modo depuración de una aplicación.

9. ¿Existe alguna manera que permita deshabilitar los puntos de interrupción? ¿Desde dónde hay que invocarla?

10. Tome nota de las barras de herramientas que se utilizan para manejar los puntos de interrupción, y estudie de qué manera habilitarlas y deshabilitarlas para su uso.

Aplicaciones ASP.NET

Más de quince años han pasado desde que la Web se instaló entre nosotros, y es por eso que cada vez más aplicaciones de escritorio se incorporan dentro de los navegadores. Incluso, la mayoría de las compañías del mundo ya comprende que, tarde o temprano, todas sus aplicaciones de gestión terminarán corriendo en los browsers. Visual Basic 2010 incluye el soporte necesario para desarrollar aplicaciones web que cumplan con todos los requerimientos de los usuarios.

QUÉ ES UNA APLICACIÓN WEB

Junto con el nacimiento de los sistemas operativos para computadoras personales, llegaron las aplicaciones que volvían útiles a estas máquinas. Procesadores de textos, planillas de cálculo y pequeñas bases de datos, entre otras más, permitieron aprovechar al máximo el uso de una computadora.
Esta situación mejoró aún más con la llegada de los sistemas operativos como Windows, que permitieron correr más de una aplicación a la vez. Luego llegaron las redes **LAN** y, por último, Internet. Si bien la gran Red de redes en un principio era lenta y en su mayoría estática, el avance de las tecnologías y la visión de futuro de muchos gurúes informáticos ayudaron a mover el motor de los avances tecnológicos en torno a Internet. Microsoft comprendió que el mundo web necesitaba una herramienta de calidad, del mismo estilo que Visual Basic, y decidió crear ASP.NET. Esta es la parte del framework .NET que permite desarrollar aplicaciones basadas en Internet. Para todos los programadores que quieren adaptar sus conocimientos al mundo de hoy, esta herramienta fue la más revolucionaria creada por Microsoft, dado que conservó la simpleza de desarrollo de aplicaciones **RAD**, a la cual la empresa ya tenía acostumbrados a todos los desarrolladores, junto con la integración de servicios web y conexión a bases de datos. ASP.NET integra dos tecnologías diferentes pero estrechamente relacionadas: los **Servicios Web** y los **Web Forms**. En el apartado **Instalación paso a paso** del **Capítulo 1** recomendamos instalar con Visual Basic 2010 la herramienta **Visual Web Developer 2010**, ubicada también en la imagen de instalación descargada de Internet. Si aún no lo hicimos, debemos realizarlo ahora, dado que es la herramienta con la cual trabajaremos durante todo este capítulo.
Si los desarrollos que estamos llevando adelante con este libro, se hacen con alguna versión Professional o Superior, y no con la versión Express, no debemos preocuparnos, dado que Visual Web Developer 2010 no existe como aplicación independiente, dado que el desarrollo de sitios o proyectos web se lleva a cabo directamente desde el **IDE** de **Visual Studio 2010**. Visual Web Developer 2010 sólo nació como una aplicación adicional para desarrollar sitios web con versiones de prueba o Express en la Suite Visual Studio.



DESARROLLOS CON WDS EXPRESS

Al igual que con las versiones pagas comerciales de Visual Studio, **Visual Web Developer Express Edition** nos permite desarrollar un sitio web completo, incluyendo el uso de bases de datos. Para conocer mejor los alcances y tipos de desarrollo que podemos lograr con él, siempre es conveniente leer el **CLUF** incluido con el producto.

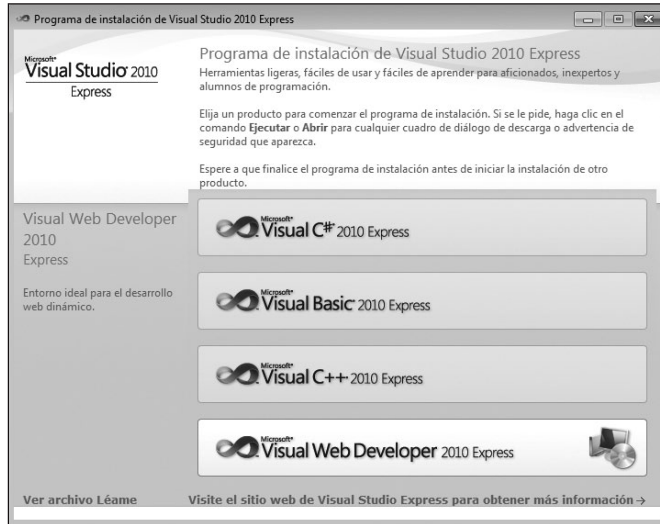


Figura 1. *Desde el disco de instalación de Visual Studio 2010 podemos instalar la aplicación Visual Web Developer.*

Las aplicaciones ASP.NET están basadas en la Web, con lo cual necesitamos un servidor apropiado para correrlas. Si nuestros desarrollos web serán instalados sobre Internet, tenemos en el mercado una amplia variedad de proveedores de hosting que ofrecen planes basados en **IIS** (*Internet Information Services*), para que todos nuestros desarrollos web se ejecuten sin problema alguno. Internet Information Services es un servidor web que se instala en computadoras que corren **Windows Server**. Las primeras versiones de IIS fueron creadas para ser ejecutadas tanto en **Windows 9x** hasta **XP profesional**, como **en Windows NT Server**, **2000** o **2003 Server**.
Desde las últimas versiones, Microsoft solo permite correr IIS en Windows Server, a raíz de diferentes problemas de compatibilidad que sufría entre las versiones de escritorio y servidor de Windows. De todos modos, podremos apreciar que cada ejercicio ASP.NET que realicemos puede probarse en nuestro entorno de escritorio sin ningún problema.
Visual Studio Express o superior incluye un servidor web que se ejecuta de manera automática cada vez que corremos nuestro proyecto. De esta manera, nos



INTERNET INFORMATION SERVICES

IIS nace en 1996 como servidor web para Microsoft Windows NT Server 3.51 y 4.0 (Server y Workstation). La idea fue transformar una PC o servidor, en un servidor web. Integrado al **Panel de control** de Windows, era posible definir los directorios principales y alias para cada subdirectorio. A partir de la versión 3.0, IIS comenzó a soportar ASP.

aseguramos de poder probar y ver funcionar correctamente los desarrollos que hagamos a lo largo de este capítulo. Para nuestro caso particular, no necesitamos instalar IIS, a menos que queramos probar nuestros desarrollos directamente en un servidor web, siempre y cuando también dispongamos de un Windows Server que permita alojar **Internet Information Services**.

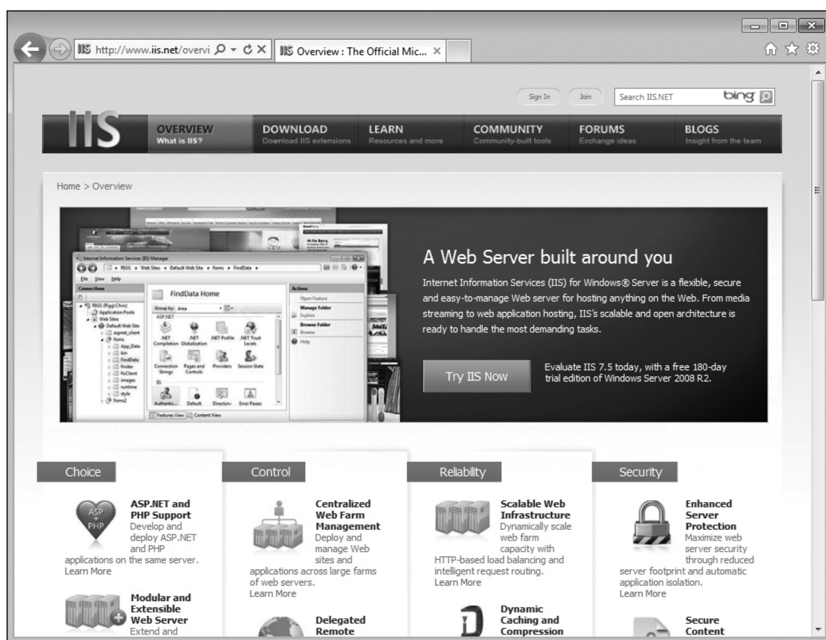


***Figura 2.** En **www.iis.net** podemos encontrar todo lo que necesitamos saber sobre **Internet Information Services**, como así también descargar su última versión.*

Concepto de Web Forms

Al instalar el framework .NET, se registran extensiones de archivos nuevas en Internet Information Services. Una de ellas es **.ASPX**, la cual identifica una de las páginas denominada **Web Form**. Los **Web Forms** son aplicaciones con una interfaz de usuario basada en la Web, a las cuales se puede acceder mediante un browser. Dichas páginas son la evolución de lo que se conocía antiguamente como **Active Server Pages** (**ASP**). La funcionalidad de este mecanismo es que, cuando un explorador web, denominado cliente, solicita un archivo con la extensión **.ASPX**, se carga de manera automática en memoria el componente **asp-net_isapi.dll** para procesar dicha solicitud. Dado que ASP.NET utiliza una extensión distinta de la antigua ASP, ambas pueden convivir dentro del mismo servidor IIS. Al igual que sucede con las aplicaciones **XAML**, las páginas **.ASPX** pueden crearse con un simple editor de textos, como el **Bloc de Notas**, pero en la era de los diseños visuales, Microsoft provee de todo lo necesario para que nuestros desarrollos sean catalogados como **RAD**.

Ejemplo de Web Forms

Conozcamos a continuación el ambiente con el cual vamos a trabajar, y luego realicemos nuestras primeras pruebas. Para hacerlo, vamos a **Inicio/Todos los programas/Microsoft Visual Studio 2010/Microsoft Visual Web Developer 2010**. Es muy probable que, si no lo hemos hecho antes, se nos pida un **Serial Key**, con lo cual, al igual que lo hicimos con **Visual Basic 2010** y que haremos en el Apéndice que trata **Visual Studio for Windows Phone 7**, deberemos registrarnos en la página de Microsoft para obtenerlo.



Figura 3. *Ingresando el número de serie obtenido en la web de Microsoft, ya podemos comenzar a trabajar con VWD 2010.*

Cabe recordar que deberemos contar con una dirección de correo válida de **Hotmail.com**, **Live.com** o un **Windows Live ID**. Si disponemos de una dirección de Hotmail o de una cuenta de Windows Live Messenger, podremos utilizarla para registrarnos. Obtenido el número de serie, lo ingresamos en la ventana que muestra la **Figura 3** y continuamos con la carga de la aplicación. Al igual que Visual Basic 2010, el IDE de Visual Web Developer 2010 Express (a partir de ahora **WDS**) se asemeja bastante en cuanto a estructura de menús, barras de herramientas, página principal, explorador de soluciones, menú de propiedades y explorador de base de datos, entre otros. Si elegimos no cerrar la página principal, siempre podremos iniciar un nuevo proyecto, o cargar uno ya creado desde



SERVIDOR WEB APACHE

Dentro de los primeros servidores web que aparecieron en el mercado se encuentra Apache. Este se basa en código abierto comúnmente usado en plataformas UNIX. También tiene su versión para correr en Microsoft Windows y Apple Macintosh, pero hasta el día de hoy ASP.NET no tiene soporte para ejecutar aplicaciones sobre este servidor.

esta pestaña, en el lado izquierdo de la pantalla. Presionamos a continuación sobre la opción **Nuevo Sitio Web**. Se abrirá el cuadro de diálogo a través del cual dispondremos de varias opciones para iniciar un nuevo proyecto con **WDS 2010**. Entre las opciones disponibles, podemos encontrar la creación de **Sitio Web ASP.Net**, **Sitio Web vacio ASP.Net**, **Sitio de entidades de datos dinámicos ASP.Net**, **Sitio web LINQ to SQL de datos dinámicos** y **Servicio WCF**.

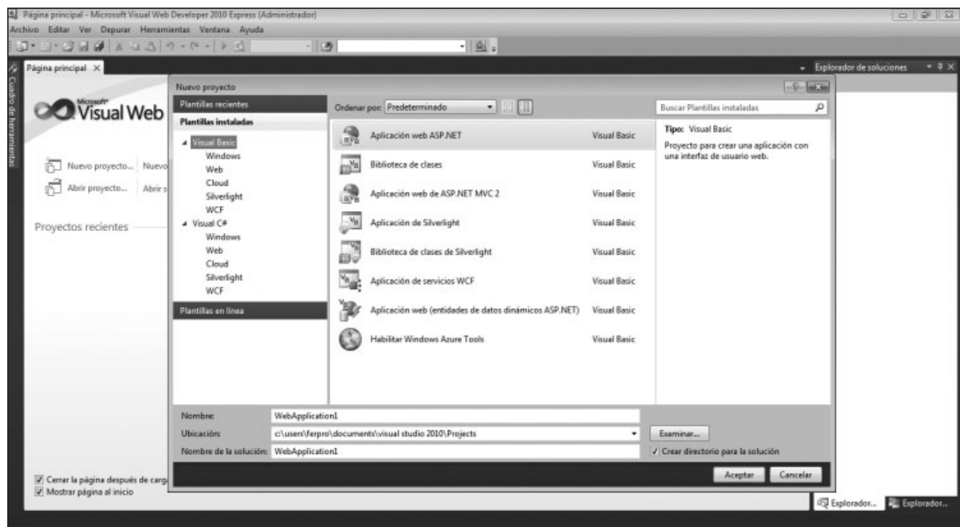


***Figura 4.** Vista de los diferentes proyectos que podemos crear con WDS 2010.*

Veamos, a través de la **Tabla 1**, qué diferencias hay entre crear un sitio web y una aplicación web en Visual Basic 2010.

ÁREA	PROYECTOS DE APLICACIONES WEB	PROYECTOS DE SITIOS WEB
Archivo de proyecto	Un proyecto de Visual Studio (.CSPROJ o .VBPROJ) guarda la información sobre el proyecto. También, la lista de archivos incluidos en este y cualquier referencia a otros proyectos.	En este caso no existe ningún archivo de proyecto (.CSPROJ o .VBPROJ). Todos los archivos de estructura de carpetas son incluidos automáticamente en el sitio.
Compilación	El código fuente se compila explícitamente en el equipo que se usa para el desarrollo. De manera predeterminada, la compilación de los archivos de código, a excepción de archivos .ASPX y .ASCX, genera un ensamblado único.	Normalmente, ASP.NET compila el código fuente de forma dinámica, en el servidor, la primera vez que se recibe una solicitud, luego de haber instalado o actualizado el sitio. Puede precompilar el sitio. De manera predeterminada, la compilación genera múltiples ensamblados.
Namespaces	Los namespaces de nombre explícitos se agregan a las páginas, controles y clases de forma predeterminada.	Los namespaces de nombre sexplícitos no se agregan a las páginas, controles ni clases de forma predeterminada, aunque sí se pueden agregar manualmente.

ÁREA	PROYECTOS DE APLICACIONES WEB	PROYECTOS DE SITIOS WEB
Implementación	Copiar el ensamblado a un servidor. El ensamblado mismo se genera al compilar la aplicación. En VS 2010 se proporcionan herramientas que integran la implementación web de IIS para automatizar las tareas de implementación.	Copiar los archivos de código fuente de la aplicación a un equipo que corra IIS. Si el sitio se precompila en un equipo de desarrollo, copiar los ensamblados generados por la compilación dentro del servidor IIS. VS 2010 proporciona herramientas para implementación, aunque estas no automatizan las tareas de implementación como las disponibles para los proyectos de aplicación web.

Tabla 1. *Resumen de diferencias entre sitio web y aplicación web, provisto por Microsoft en su página de MSDN:* ***http://msdn.microsoft.com/es-es/library/dd547590.aspx****.*

Para comenzar a comprender mejor el fascinante mundo del desarrollo web, iniciaremos una introducción a la creación de un **Web Form**. Con este fin, crearemos un nuevo proyecto desde el menú **Archivo/Nuevo Sitio Web**, al que nombraremos **ProyectoWebForm**. Como directorio de almacenamiento, elegimos el que nos ofrece por defecto Visual Studio. Veamos en la **Figura 5** cómo queda confeccionada la estructura de nuestro proyecto.

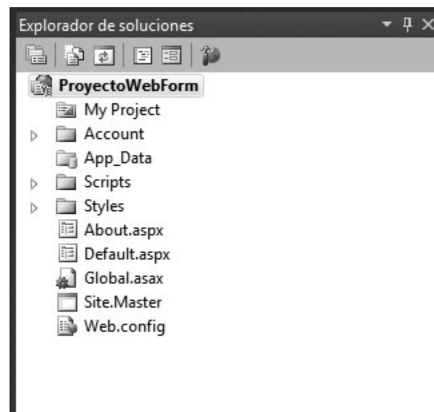


Figura 5. *El cuadro* ***Explorador de Soluciones*** *contiene el nombre de nuestro proyecto, las carpetas y archivos que lo componen.*



OBLIGACIONES DE ASP.NET

En **HTML** y en **ASP**, dentro de una misma página se soportaba más de una etiqueta **<form>**. **ASP.NET** solo permite una única etiqueta **<form>** por página. Para el caso de que se deba manejar más de una etiqueta de manera irrevocable, habrá que utilizar una página HTML con etiquetas **<frame>**, aunque estas ya no se emplean desde hace muchos años.

Desde el cuadro **Explorador de Soluciones** podemos ver que el proyecto ya incluye archivos básicos, como la página web principal, denominada **Default.aspx**; la carpeta **Styles**, que aloja los archivos de estilos **CSS**, encargados de permitir darle un "look n' feel" a nuestra aplicación web; la carpeta **Account**, que crea por defecto los archivos que permitirán integrar en nuestra aplicación **un Sistema de Registración**, **Login**, **Cambio de password** y **Restaurar password**. También encontramos una carpeta **App_Data**, donde se alojará la o las bases de datos que podemos llegar a utilizar en un desarrollo. Este mismo cuadro contiene una serie de iconos en su extremo superior, desde los cuales podemos, por ejemplo: ver la hoja de propiedades de nuestro proyecto, el primer icono denominado **Propiedades** y el último icono de la fila que nos lleva a la página de configuración ASP.NET. Si hacemos clic en este último, se abrirá una ventana de nuestro navegador web por defecto en la cual encontraremos una aplicación web llamada **Herramienta de Administración de Sitios Web**. Desde esta herramienta podemos configurar la seguridad de nuestro sitio, la página principal y parámetros adicionales de nuestra aplicación, como la **Configuración del correo SMTP** para el envío de **e-mails Forms**, u otros correos electrónicos a nuestra casilla de correo, detener o iniciar el servicio web para nuestra aplicación, la seguridad ante ataques internos o externos hacia nuestro sitio web y otras opciones.



Figura 6. *Configuración de las propiedades del sitio web. La dirección ejecutada es una dirección local de un servicio de servidor web que VWD 2010 levanta en la computadora.*

Seleccionamos en el **Explorador de Soluciones** el nombre de nuestro proyecto, y presionamos el botón **Propiedades** o la combinación de teclas **ALT + Enter**. Accederemos a la hoja de propiedades desde la cual podemos administrar las propiedades, la configuración y los recursos que integran el proyecto. Allí veremos que las distintas configuraciones se separan en fichas laterales, ubicadas sobre el lado izquierdo de la página. La información que encontramos se almacena de manera automática al cambiarla. Las páginas y la función que componen las propiedades de nuestro proyecto se describen en la **Tabla 2**.

PÁGINA	DESCRIPCIÓN
Aplicación	Contiene la configuración que describe el tipo de aplicación y su comportamiento, los objetos de inicio e información de ensamblado.
Compilar	Contiene las instrucciones básicas de compilación de nuestro proyecto.
Referencias	Contiene la lista de componentes, referencias y espacios de nombre que se utilizan en el proyecto.
Web	Permite cambiar la página de inicio, seleccionando otra del proyecto. También permite configurar la ejecución de programas externos y la configuración de directorios de trabajo.
Empaquetar / Publicar	Permite aplicar una configuración específica para la creación de nuestro proyecto.
Empaquetar / Publicar SQL	Permite configurar una configuración específica sobre la base de datos SQL Server que compone nuestro proyecto (si es que la hay).
Aplicaciones de Silverlight	Permite configurar y agregar aplicaciones desarrolladas con Silverlight.
Recursos	Permite configurar determinada información que alimenta a nuestro proyecto.
Configuración	Permite almacenar y recuperar configuración de determinadas propiedades para usar en nuestra aplicación. Esta información puede separarse por usuario, con lo cual es posible mantener configuraciones distintas para cada uno de ellos.
Firma	Permite configurar una firma que valide nuestro proyecto como un proyecto certificado.
Extensiones My	Permite configurar las extensiones My de los espacios de nombre creados en el proyecto.

***Tabla 2.** Las propiedades de nuestro proyecto, detalladas según la pestaña. Es posible encontrar una descripción más completa de este tema en **http://msdn.microsoft.com/es-es/library/bb1aa8f1.aspx**.*



WEB SERVICES

Un servicio web agrupa un conjunto de protocolos que permiten intercambiar datos entre aplicaciones, sin importar el lenguaje de programación en el que hayan sido desarrolladas. Los servicios web son ideales para intercambiar datos entre redes de computadoras a través de Internet, gracias a los estándares abiertos regidos por los comités OASIS y W3C.

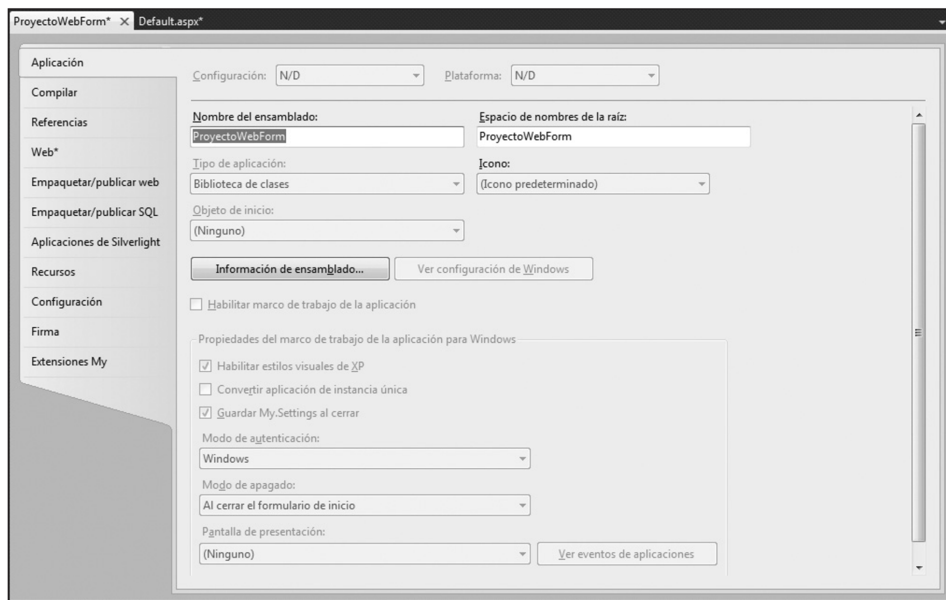


Figura 7. *Página de propiedades de nuestro proyecto.*

Antes de iniciar el desarrollo de nuestro primer ejemplo, debemos conocer qué función cumple el archivo **default.aspx**. Este archivo será el que, por defecto, cargará nuestro sitio web al iniciarse. Todos los archivos con la extensión **.ASPX** de nuestros proyectos se dividen en dos partes:

- El archivo en sí, que contiene toda la estructura de la página, información de componentes, lenguaje en el que se programará cada página, etc.
- El archivo con extensión donde se guardará el código fuente perteneciente a él, que se conforma de la siguiente manera: **nombredearchivo.aspx.vb**.

Si bien hay una manera de conservar un archivo único que mantenga el diseño de la página web y el código que escribiremos, siempre es conveniente separarlos, dado que será mucho más cómodo que tenerlos todos juntos. Veamos a continuación rápidamente cómo se conforma un archivo **.ASPX**:



MÁS PESTAÑAS DE PROPIEDADES

Según la versión de Visual Studio 2010 que estemos utilizando, es probable que la cantidad de pestañas que componen la página de propiedades de un proyecto varíe. Para las versiones comerciales de **VS 2010** existen otras pestañas con más configuraciones. En la **URL** detallada al final de la **Tabla 2** podemos verlas todas.

```
<%@ Page Title="Página principal" Language="vb" MasterPageFile="~/Site.Master"
AutoEventWireup="false"
    CodeBehind="Default.aspx.vb" Inherits="ProyectoWebForm._Default" %>

<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="Head
Content">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainCon
tent">
    <h2>
        ASP.NET
    </h2>
    <p>
        Para obtener más información acerca de ASP.NET, visite <a
href="http://www.asp.net"
title="Sitio web de ASP.NET">www.asp.net</a>.
    </p>
    <p>
        También puede encontrar <a
href="http://go.microsoft.com/fwlink/?LinkID=152368"
            title="Documentación de ASP.NET en MSDN">documentación sobre
ASP.NET en
MSDN</a>.
    </p>
</asp:Content>
```

Analicemos el contenido de una página **.Aspx**. En principio, se declara la información básica de cada página, como el título; **Page Title="Página principal"** indica el lenguaje con el cual se programa dicha página web; **Language="vb"** señala cómo debe estar compuesta estéticamente dicha página; **MasterPageFile="~/Site.Master"** determina en qué archivo se almacena el código fuente de la página; **CodeBehind="Default.aspx.vb"**, y algunas otras funciones más que veremos en el transcurso de este capítulo. Toda página web **.ASPX** se puede observar y trabajar de dos maneras: en modo **Código**, editando y escribiendo directamente el código que estructurará el diseño de la página; y mediante el modo **Diseño**, que nos permitirá ser más productivos, sobre todo, si tenemos que agregar y configurar controles web. Por último, encontramos un estado intermedio, que separa el área de trabajo en dos partes: el **Modo Dividir**, que será la opción elegida para comenzar nuestro proyecto. En la vista **Código** de nuestra página **Default.aspx**, vamos a borrar todo el código cargado, menos la primera línea, donde se declara declaran los

datos relevantes de la web. El siguiente **Código** es el que debe quedar en el modo código de **default.aspx**:

```
<%@ Page Title="Página principal" Language="vb" MasterPageFile=""
AutoEventWireup="false"
     CodeBehind="Default.aspx.vb" Inherits="ProyectoWebForm._Default" %>
```

En la referencia **MasterPageFile** eliminamos la opción que nos sugiere el sitio web, y dejamos vacía la referencia a la página maestra. Deben quedar las comillas dobles sin contenido de referencia, como figura en el código anterior. A continuación, visualizamos nuestro **Cuadro de herramientas**, y agregamos un control **TextBox**, un control **Button** y un control **Label** a **default.aspx**. La página debe quedar conformada como vemos en la **Figura 8**.

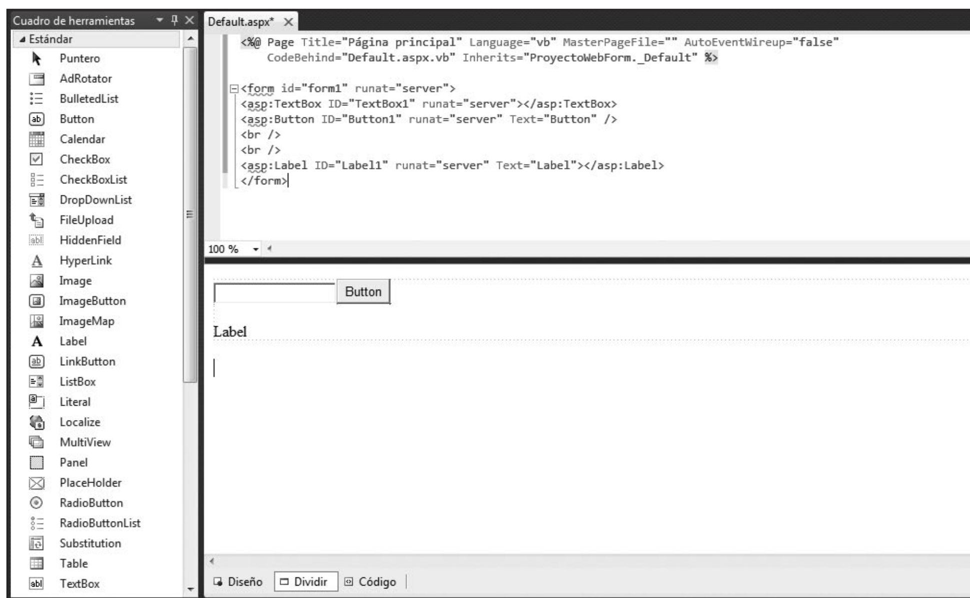


***Figura 8.** Página Default.aspx en visualización mixta.*



VISUALIZACIÓN DE PÁGINAS ASP.NET

La suite Visual Studio incluye dos maneras de visualizar las páginas ASP.NET y el código. La primera es ver la vista de diseño como un archivo y el código por separado en otro archivo, que se llama igual que la página, más la extensión **.VB** o **.CS** al final. La otra es poder visualizar tanto el diseño como el código en una sola página.

Antes de escribir una sola línea de código, podemos probar qué tal se verá nuestro proyecto en el navegador. Para hacerlo, nos aseguramos de guardar todo en primera instancia y, luego, presionamos la tecla **F5**. Se iniciará por defecto una instancia de web server que cargará nuestro proyecto y, a continuación, el navegador predefinido que tengamos en la computadora.

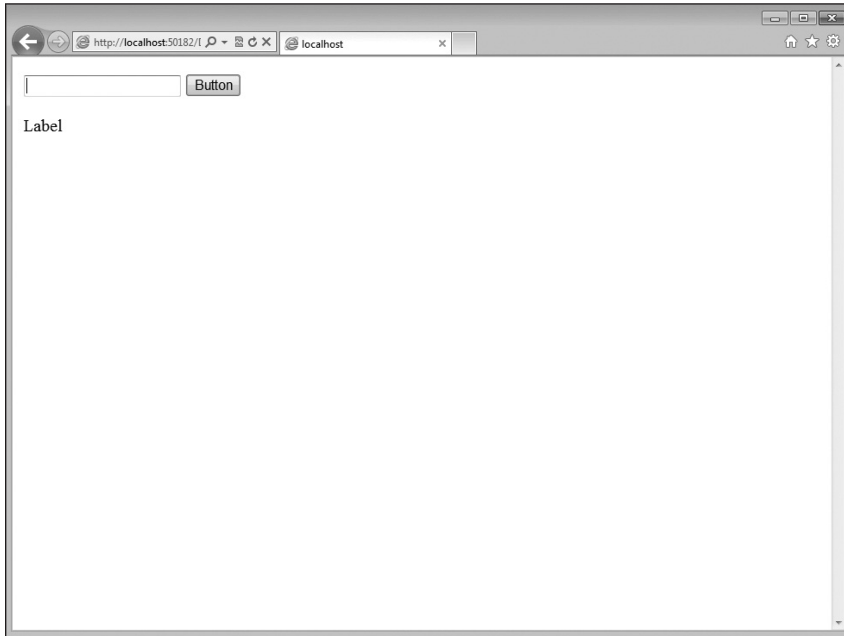


Figura 9. *Nuestra página principal corriendo en el navegador web.*

Cerremos el navegador para seguir trabajando sobre nuestro proyecto. Si seleccionamos uno de los controles agregados en la página, veremos que el código fuente relacionado a dicho control se pinta de un **color gris claro**. Esto nos ayuda a identificar el código rápidamente para aplicar las modificaciones sobre el control de manera visual o del código. Vamos a agregar nuestro nombre en el control **TextBox**. Podemos hacerlo editando las propiedades de dicho control, al igual que lo hacíamos en cualquier proyecto **Windows Forms**, o escribiendo la propiedad en la **Vista Código** de nuestra página. Vamos a hacerlo de esta última manera. La línea de código perteneciente al control **TextBox1** debe quedar así:

```
<asp:TextBox ID="TextBox1" text="Fernando" runat="server"></asp:TextBox>
```

Notaremos que, entre la línea divisoria de las vistas de nuestra página **.ASPX** puede aparecer una leyenda que indica **"La vista código y la vista Diseño no están sincronizadas. Haga clic aquí para sincronizar las vistas"**. Podemos hacer clic directamente

sobre la leyenda o, de lo contrario, presionar el botón **Guardar** de la barra de herramientas. De este modo, quedarán ambas vistas sincronizadas. Acto seguido, escribimos el siguiente código dentro del evento **clic** de **Button1**. Para lograrlo, hagamos doble clic sobre la vista **Diseño** de **Button1** y entraremos directamente al **Editor de códigos**. Escribimos lo siguiente en el evento **Clic**:

```
Label1.Text = “Bienvenido “ & TextBox1.Text & “. Esta es su primer página
web.”
```

Al igual que para la creación de aplicaciones Windows, Visual Basic permite escribir el código necesario para que nuestro proyecto web funcione, con algunas salvedades. El código que escribimos funcionará, pero si borramos nuestro nombre del **textBox** y luego presionamos el botón, el mensaje quedará inconsistente. Para esto, modificamos este código antes de probarlo:

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles
Button1.Click
    If TextBox1.Text.Trim = “” Then Label1.Text = “Debe ingresar un texto
válido en textBox1
para continuar...” : Exit Sub
    Label1.Text = “Bienvenido “ & TextBox1.Text & “. Esta es su primer
página web.”
End Sub
```

Al presionar **Button1**, si no hay ningún texto ingresado, el programa mostrará una sugerencia a través de **Label1** para que se ingrese un texto válido, luego de lo cual saldrá de la ejecución del código. Si no, directamente mostrará en **Label1** el saludo con nuestro nombre. Guardamos a continuación el proyecto y lo ejecutamos. Probemos entonces el correcto funcionamiento del código escrito hasta aquí.



WSDL

El WSDL (*Web Services Description Language*) es un formato XML utilizado para intercambiar datos en los servicios web a través de una interfaz pública. WSDL se combina generalmente con SOAP y XML Schema, los cuales, mediante un programa cliente que se conecta a un servicio web, permiten leer correctamente el WSDL y obtener las funciones disponibles en el servidor.



***Figura 10.** Nuestro programa en ejecución advierte que no hay texto en **TextBox1**, y le notifica al usuario.*

Ahora ingresamos nuestro nombre en el **TextBox1** y volvemos a presionar **Button1**.



***Figura 11.** Ahora sí, nuestra **Web App** nos saluda correctamente.*

Ahora vamos a hacer que nuestro sitio web tome el formato predeterminado en **Site.Master**. Para esto, modificamos directamente el código fuente de **Default.aspx** para que quede de la siguiente manera:

```
<%@ Page Title="Página principal" Language="vb" MasterPageFile="~/Site.Master"
AutoEventWireup="false"
    CodeBehind="Default.aspx.vb" Inherits="ProyectoWebForm._Default" %>

    <asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainCon
tent">

<asp:TextBox ID="TextBox1" text="Fernando" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Button" />
<br />
<br />
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
</asp:Content>
```

Site.Master es un archivo que oficiará de plantilla para todo nuestro sitio web. Si deseamos mantener el mismo diseño en toda la web, ya sea en el encabezado o en el pie de página, los menús superiores o laterales u otros objetos predeterminados de nuestra web, conviene editar este archivo, formateándolo para que nos garantice que todo el contenido del website lucirá de la misma manera en cada una de las páginas que lo compongan. Logrado esto vamos a poder disfrutar de un sitio web homogéneamente parejo en cuanto a diseño y estructura de menús y botones de información o links hacia otras secciones. Esta característica destaca a ASP.Net por sobre el resto de las herramientas que permiten crear un sitio web estéticamente parejo con casi nada de esfuerzo por parte de un diseñador o desarrollador.



MASTER PAGE O SITE MASTER

Master Page es la definición correcta para definir una página maestra que contendrá todo el diseño que el sitio web deberá respetar. Site Master es el nombre del archivo donde se aloja la definición principal de Master Page. Esta se puede combinar, a su vez, con hojas de estilo CSS para otorgarle un mejor "look" a nuestro sitio web.

Página de contenidos

Page Content
Contenido principal
del sitio web
Esto es una página específica que estará incluida dentro de Master Page para conformar el sitio web completo.

Master Page

Header Region

Toolbar

Page Content

Footer region

Página ASP .Net
que ve el usuario

Header Region

Toolbar

Page Content
Contenido principal
del sitio web
Contenido fusionado y compilado para mostrar al usuario que navegará nuestra web.

Footer region

***Figura 12.** Este diagrama muestra cómo se fusiona una página web con una **Master Page** para dar vida a páginas estéticamente iguales para nuestro sitio, sin mucho esfuerzo.*

Como vimos al inicio de la explicación de **Default.aspx**, **Site.Master** se declara en la primera etiqueta de cada página, como plantilla para referencia de cada página web.



LA WEB 2.0

Este término comenzó a regir en el año 2004, rebautizando a los portales de Internet que permitían sociabilizarse más con el usuario, e intercambiar información y compartir la que fuera publicada en los sitios web. ASP.NET permite tranquilamente crear aplicaciones web 2.0 sin necesidad al igual que cualquier otro lenguaje de programación web.

Probemos cómo quedó nuestro sitio web mejorado aprovechando el look predeterminado en la plantilla **Site.Master**. Ejecutamos el proyecto con **F5**.

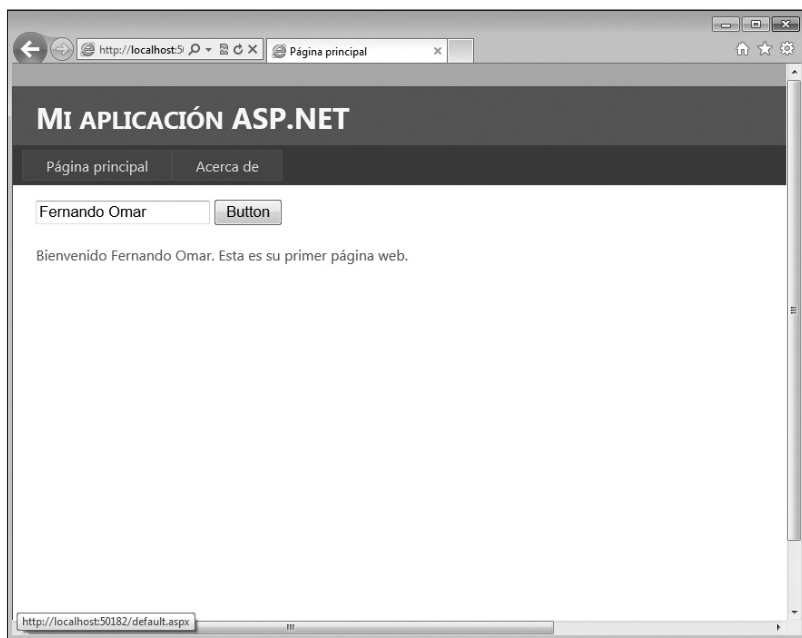


Figura 13. *Default.aspx ya tomó el look n' feel de Site.Master para lucir una página web más estilizada.*

Al igual que con **Windows Forms**, cada página **.ASPX** tiene un evento denominado **Load()** que puede ejecutar código cuando se carga la página. Vamos a modificar la página en la cual estuvimos trabajando para que los controles cambien sus propiedades y valores. Presionamos el botón derecho del mouse sobre la página en la cual trabajamos y seleccionamos, del menú contextual, la opción **Ver Código**. A continuación, agregamos lo siguiente:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    TextBox1.Text = “Fernando Luna”
    Label1.Text = “”
    Button1.Text = “Saludar”
End Sub
```

En el código agregado podemos ver que estamos predeterminando que el valor que aparecerá en **TextBox1**. **Label1** no tendrá ningún texto hasta tanto lo escriba la acción clic de **Button1**, cuyo texto cambió por **Saludar**.

Web Forms dinámicos

Veamos a continuación qué acciones se ejecutan en el servidor y cuáles en el cliente cuando un **explorador web** interactúa con un **web form**. **ASP** tenía un problema grande, ya que no podía separar la interfaz de usuario, del código de la aplicación en sí, con lo cual un diseñador gráfico tenía que interactuar sí o sí con un programador web, y un programador web tenía que aprender necesariamente el trabajo de un diseñador. A ningún programador nunca le atrajo la idea de tener que manipular temas relacionados con interfaces de usuarios. Actualmente, cuando los navegadores solicitan un archivo **.ASPX**, una **DLL** de **ASP.NET** recibe la solicitud, carga el archivo y busca en él información sobre el atributo **Inherits**. El valor contenido en este atributo es el nombre de la clase contenedora del código que asociará los sucesos por ejecutar del lado del servidor para esta página. Un claro ejemplo sería hacer clic sobre un botón, que conecta con una base de datos, obtiene valores de una tabla o vista y rellena un **DataGridView** para mostrar el resultado. Todas estas acciones se ejecutan del lado del servidor, donde se alojan la base de datos, las tablas y el **DataSet** que se llenará para, luego, cargar los datos en la grilla. Para mejorar los principales problemas que tenía ASP original, ASP.NET decidió hacer algunos cambios más que interesantes en la estructura de trabajo de su servidor **IIS**. En principio, se optó por que cada llamado a la DLL para compilar páginas no sea procesado por la librería original, sino por una copia de ella. A esto se lo conoce como **Shadow Copy**, y tiene la ventaja de que, si hay que compilar varias veces la DLL por cambios que debemos realizar en ella, no debemos parar el servicio de IIS para copiar la nueva DLL y proseguir con su uso.

```
<%… Inherits=”ProyectoWebForm._Default” %>
```

Toda página web tiene un ciclo de vida establecido, y esto no es excepción para las **.ASPX**. Como vimos anteriormente, los controles utilizados en las páginas ASP.NET tienen un funcionamiento similar a los **Windows Forms**, ya que disponen de casi los mismos sucesos y eventos. Como los controles **Button** disponen del suceso **Click**, los controles **textBox** disponen del suceso **TextChanged** y los **ListBox** disponen del suceso **SelectedIndexChanged**. Así sucesivamente para cada control equivalente entre una plataforma y otra.

La clase Page

La clase **Page** es equivalente en ASP.NET a la clase **Form** de **Windows Form**, con lo cual se dispone de sucesos **Load** y **Unload**. Aun así, aunque el modelo activador por sucesos es similar, debemos destacar que ambos mundos siguen siendo completamente distintos. En **Windows Forms**, los sucesos se activan cuando el usuario actúa

de manera directa con los sucesos de uno o más objetos. En ASP.NET los sucesos se lanzan dentro del servidor y no dentro del explorador, con lo cual las acciones de los controles se ejecutan cuando el formulario envía una petición hacia el servidor, lo que se conoce en el mundo web como **POST** o **GET**, eventos que se disparan cuando un usuario hace clic en el botón **SUBMIT** de un **Web Form**. Analicemos a continuación los pasos que ocurren cuando se envía un **Form** hacia el servidor:

- ASP.NET carga la clase que está detrás del código y se activa el suceso **Page_Init**. En este momento, el código no puede determinar si es la primera vez que se solicita el **form** o no, y tampoco puede recuperar automáticamente los valores que el usuario escribió en los controles.
- Luego de que **Page** y los controles fueron inicializados, se activa el suceso **Page_Load**. Allí se puede determinar si la página ya estaba en ejecución o no, para saber si debe devolver datos o no (esto se realiza mediante la propiedad **IsPostBack**).
- Los sucesos que se relacionan con los controles se activan en ese momento, siempre y cuando el proceso sea una devolución de datos.
- El último suceso que se lanza es el que se origina por la acción de devolución de datos, generalmente, el suceso **Click** del control **Button**.
- El objeto **Page** lanza el suceso **Unload** cuando se espera que se liberen todos los recursos, y se cierren los archivos y conexiones a bases de datos, si es que existían.

El objeto **Web Forms** lanza los sucesos analizando el estado en el que se encuentra cada control del formulario. Los sucesos son enviados de regreso al servidor y son comparados con los estados de los controles en el momento en que el formulario fue enviado al explorador web. Este proceso se lleva a cabo gracias al campo oculto **_VIEWSTATE**. Si, por ejemplo, se lanza un suceso **TextChanged** propio de un **textBox**, es porque el control contiene un valor en su cadena de caracteres distinto del que tenía en su valor original. Esto ocurre aunque el usuario haya modificado reiteradas veces el valor del control antes de enviar el Form de vuelta al servidor. Cada control **Web Form** lanza una menor cantidad de sucesos que los equivalentes a **Windows Form**. Uno de los motivos es porque los controles **Web Form** no disponen de sucesos como **MouseEnter**, **MouseExit**, **GotFocus** y **LostFocus**, dado que estos son utilizados para retroalimentar de manera instantánea los formularios y proyectos desarrollados en **Win32**.



COMPILACIÓN BAJO DEMANDA

Visual Studio .NET puede funcionar correctamente aunque la DLL compilada que se ejecuta en el Explorador de la página no esté creada. Esto se conoce como compilación bajo demanda. Para que esto ocurra, **ASP.NET** debe comprobar la extensión del archivo de código asociado a las paginas **.ASPX**, para decidir qué compilador utilizar: **VB**, **C#** u otro asociado al lenguaje.

Propiedades

Hay dos propiedades importantes que pueden servirnos para manejar la carga de las páginas web y la activación o ejecución o no de sus controles o determinados eventos. Según lo que vinimos hablando hasta este momento acerca de los métodos de ejecución y actuación de cada control, el evento **Page_Load** se ejecuta tanto en la primera carga de la página en el navegador web, como cada vez que se ejecuta el evento **SUBMIT** por el clic en algún botón relacionado al **Web Form**. Cada control seguramente ya tenía un valor o datos asignados en pantalla, y a menos que estemos cambiando de página web, convendrá siempre mantener el que cada uno poseía. ASP.NET, a diferencia del antiguo ASP, guarda de manera predeterminada el estado de cada control por cada vez que se recarga la página. Si no necesitamos que estos valores sean almacenados, conviene desactivar el mecanismo de persistencia para un determinado control. Esto se hace estableciendo la propiedad **EnableViewState = False**.

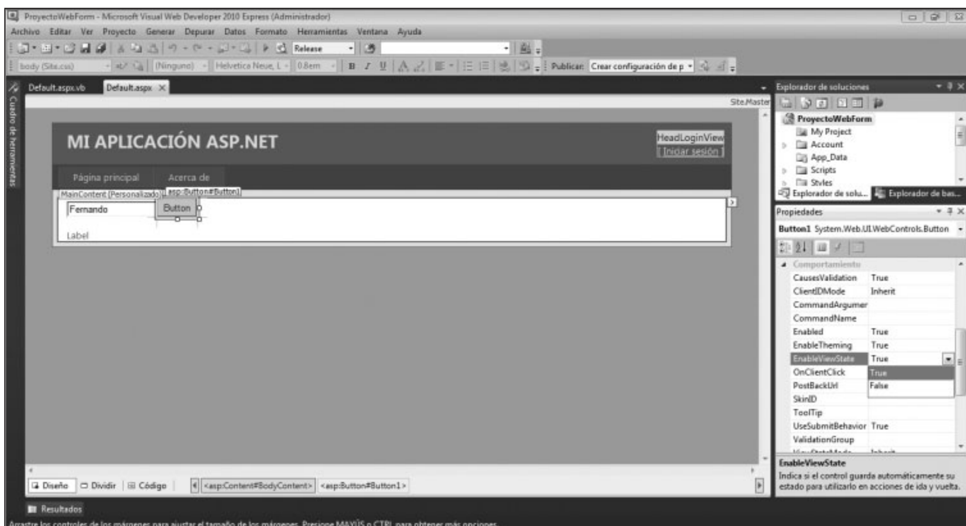


***Figura 14.** Seleccionando **Button1** y revisando sus propiedades, podemos ver que **EnableViewState** está seteado en **True**.*

Si no deseamos que ningún control conserve su estado, entonces será más fácil que configuremos la propiedad del objeto **Page** en vez de hacerlo control por control:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles
Me.Load
        Me.EnableViewState = False
            ...
```

Cuando creamos una aplicación **ASP.NET**, debemos prestar especial atención a esto. Si no necesitamos realmente tener activo y mantener el estado de los controles cada vez que se presiona un botón, convendrá desactivar la propiedad **EnableViewState**. De esta manera, viajarán menos datos hacia el servidor y, por lo tanto, vendrán menos datos desde él hacia el navegador web. Esto agilizará el funcionamiento de las **WebApps**. Otra opción para controlar la recarga de una página web es la propiedad **IsPostBack**. A continuación cambiaremos algunos controles en **Page Load**:

```
...
TextBox1.Text = "Fernando Luna"
Label1.Text = ""
Button1.Text = "Saludar"
...
```

El código anterior es el que figura en el evento **Load**. Si presionamos **Button1**, la máquina por defecto saludará a Fernando Luna. Ahora, si ponemos **Nicolás Luna** en el **textBox** y volvemos a pedir que la máquina salude, notaremos otra vez que saluda a **Fernando Luna**. Esto ocurre porque cada vez que presionamos **Button1**, se ejecuta el evento **Load**, que setea por defecto a **TextBox1.Text** tal como figura en el código anterior, pasando por alto el valor que ingresamos en el **TextBox**. Para que esto no ocurra, tenemos que modificar el código para que quede de esta manera:

```
If Not IsPostBack Then
    TextBox1.Text = "Fernando Luna"
    Label1.Text = ""
    Button1.Text = "Saludar"
End If
```

IsPostBack mantiene el valor **booleano** a través del cual sabemos si la página se carga por primera vez, o si ya estaba cargada y se estaba refrescando. Así, podemos



MANEJO DE INFORMACIÓN VÍA WEB

Las páginas web han causado una revolución total y un replanteo importante en la resolución de problemas como el tratado aquí. Antes, en ASP clásico, mantener el estado de cada componente era un verdadero dolor de cabeza, y en parte esto era lo que hacía relativamente lento al lenguaje **ASP** al lado de su principal competidor, **PHP**.

controlar que el valor ingresado en el control **TextBox1** no sea modificado por el que predeterminamos al cargar la página.

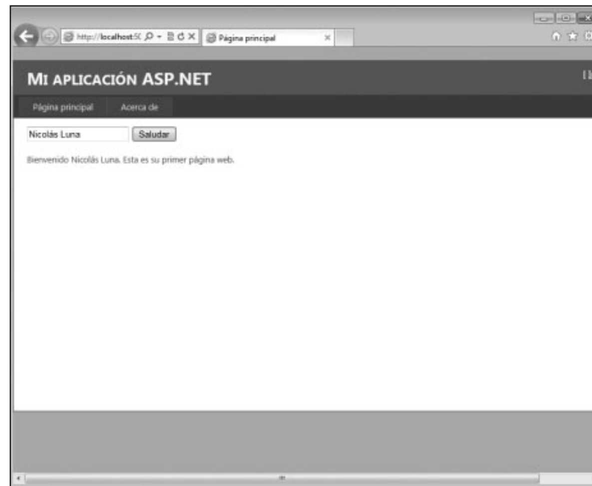


***Figura 15.** Cambiando el valor de TextBox1, vemos que ya no ocurre el problema anterior al refrescar la página.*

La clase **Page**, a su vez, tiene varias propiedades importantes que debemos conocer. Eso lo veremos a través de la **Tabla 3**.

CATEGORÍA	PROPIEDAD	DESCRIPCIÓN
Objetos ASP	Request	Devuelve el objeto HttpRequest que describe la solicitud enviada al servidor.
	Response	Devuelve el objeto HttpResponse para controlar cómo se envían datos al Explorador web cliente.
	Server	Devuelve el objeto HttpServerUtility que representa el servidor.
	Application	Devuelve el objeto HttpApplicationState para almacenar los datos compartidos entre los clientes de la aplicación.
	Session	Devuelve el objeto HttpSessionState que se utiliza para almacenar los datos del usuario en la sesión actual.
Comportamiento	ClientTarget	HttpBrowserCapabilities sirve para detectar el tipo de explorador que utiliza el cliente.
	EnableStateView	Si todos los controles utilizados en una página tienen la misma propiedad activa, toma como valor True.
	SmartNavigation	Permite configurar la aplicación para activar la navegación inteligente.
	IsPostBack	Si es False, es porque la página se carga por primera vez.
	ErrorPage	Si se produce una excepción en la página actual, dirigirá a una página de error específica.
Validación	Validators	Contiene la colección de controles de validación para la página.
	IsValid	Devuelve True si todos los controles tienen la validación de página.

CATEGORÍA	PROPIEDAD	DESCRIPCIÓN
Rastreo	Trace	TraceContext es un objeto que se puede utilizar para obtener información de rastreo y diagnóstico.
	TraceEnabled	Esta propiedad con un valor True activará el rastreo de la página.
	TraceMode	Especifica la manera de ordenarse los mensajes de rastreo (SortByCategory, SortByTime).
Otros	Controls	Colección de controles de la página.
	ViewState	ViewState conserva los valores entre mensajes de vuelta.
	TemplateSource-Directory	Devuelve el directorio virtual donde reside la página.
	Cache	System.Web.Caching.Cache es el objeto que almacena temporalmente una caché para compartirla entre los clientes de la aplicación.
	User	Devuelve un objeto iPrincipal que tiene información sobre el usuario actual. La propiedad Identity posee otras propiedades (Name, IsAuthenticated y AuthenticationType).

***Tabla 3.** Estas son las principales propiedades de la clase Page.*

Métodos y sucesos

Así como existen propiedades para la clase **Page**, también existen métodos. Veamos algunos de los principales en la **Tabla 4**.

CATEGORÍA	MÉTODO	DESCRIPCIÓN
Ruta	MapPath	Transforma la ruta virtual en física.
	ResolveUrl	Transforma una URL relativa en absoluta, utilizando TemplateSourceDirectory.
Controles	HasControls	Si la página contiene controles, devuelve True.
	FindControl(id)	Si encuentra el control, devuelve la ID especificada; si no, devuelve Nothing.
	LoadControl(ruta virtual)	Carga un control de usuario (UserControl) desde un archivo .ASCX.
	DataBind	Establece la conexión de los controles hijos con la fuente de datos.
	Validate	Fuerza a que todos los controles de validación validen a sus controles asociados. Ocurre al hacer clic en algún botón, cualquiera sea su tipo, siempre que su propiedad CauseValidation = True.

***Tabla 4.** Métodos de la clase Page.*

Veamos a través del siguiente código un breve ejemplo de cómo el método **MapPath** convierte una ruta virtual en una física:

```
Dim lector as New System.IO.StreamReader(Me.MapPath("/data/valores.dat"))

'Conocer el directorio actual
Dim ActualDir as String = Me.MapPath(".")
'Directorio principal, o Padre
```

```
Dim PrincipalDir as String = Me.MapPath("..")
'Directorio Raíz
Dim RaizDir as String = Me.MapPath("/")
```

La clase **Page** hereda varios sucesos desde **Control** y **TemplateControl**. Todos ellos reciben un objeto **EventArgs** en su segundo argumento. Entre estos sucesos recibe el objeto **Error**, que se lanza al producirse una excepción no controlada por código. Este suceso permite aclarar el error y redirigir el flujo de ejecución hacia otra página:

```
Private Sub Page_Error(…)
    Server.ClearError() 'Limpia el tipo de error acontecido
    Server.Transfer("ayuda.aspx") 'Envía al usuario a otra página.
End Sub
```

Directivas de la clase Page

Una gran cantidad de propiedades que se configuran en la ventana correspondiente de **Visual Web Developer 2010** se transforman en atributos de la directiva **@Page** en el inicio de cada archivo **.ASPX**. Algunos de estos valores aparecen también como propiedades del objeto **Page**, y pueden ser configurados mediante código. De todos modos, ocurre en algunos casos que solo pueden ser configurados en la directiva **@Page**. Los archivos **.ASPX** pueden guardar también otros tipos de directivas, como **@Import** e **@Implements**, entre otras. Debemos recordar siempre que se acepta una **Directiva @Page** por cada archivo **.ASPX**, la cual normalmente, como vimos al inicio de este capítulo, se sitúa al principio del archivo. Y cuando creamos una página, de manera automática se crea el atributo **Inherits**. La directiva **@Import** se ocupa de importar un espacio de nombres a la página **.ASPX**, con lo cual es equivalente a la instrucción **Imports** de las aplicaciones **Windows Forms**. La directiva **@Import** puede especificar solo un espacio de nombres. Si necesitamos importar varios espacios, debemos utilizar varias directivas.

```
<%@Import namespace="System.Xml" %>
<%@Import namespace="System.Xml.Xsl" %>
<%@ Page Title="Página principal" Language="vb" MasterPageFile="~/Site.Master" AutoEventWireup="false"
    CodeBehind="Default.aspx.vb" Inherits="ProyectoWebForm._Default" %>

    <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">

<asp:TextBox ID="TextBox1" text="Fernando" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Button" />
<br />
<br />
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
</asp:Content>
```

Figura 16. *Vemos aquí cómo incluimos la directiva @Import para traer referencias a namespaces.*

La directiva **@Assembly** vincula un ensamblado a la página actual en el momento de compilar el proyecto, para que todos los ensamblados estén disponibles en el código de la página. La directiva debe incluir el atributo **Name** o **Src**. No hace falta referenciar mediante esta directiva a aquellos ensamblados que estén alojados en el directorio **\Bin** de nuestra aplicación web, ya que de esta manera se vinculan automáticamente. Para usar esta directiva se debe incluir la ruta relativa al mismo:
Veamos un ejemplo de esto a través del código siguiente:

```
<%@ Assembly Name = "Functions.dll" %>
<%@ Assembly Src = "Functions.vb" %>
```

La directiva **@Register** asocia un alias con un namespace o clase para, de esa manera, permitir que el desarrollador use una notación precisa al referirse a controles de servidor personalizados:

```
<%@ Register tagprefix = "alias" Namespace = "namespace" ASsembly =
"Asmname" %>
```

La directiva **@Reference** le indica a ASP.NET que otro archivo fuente de página o control debe compilarse dinámicamente y vincularse a dicha página. A continuación, el código correspondiente a cada objeto:

```
<%@ Reference page = "otrapagina.aspx" %>
<%@ Reference control = "elcontrol.ascx" %>
```

La directiva **@Control** es usada para especificar atributos de un control específico personalizado, con lo cual sólo será utilizada solo en los archivos **.ASCX**. Soporta atributos válidos para la directiva **@Page**.
Existen más directivas, pero llevarían al menos medio manual poder explicarlas a todas.



OTRAS DIRECTIVAS

Existen otras directivas además de las mencionadas: **@Implements** indica que una página implementa una interfaz dada, lo que equivale a la instrucción **Implements** de Visual Basic .NET; en tanto que **@Control** sirve para especificar atributos de un control personalizado. Esta última solo se usa dentro del tipo de archivos .**ASCX**.

Controles Web Forms

Los controles para **Web Forms** se dividen en varios grupos. Algunos de ellos sirven para ejecutarse del lado servidor, y otros lo hacen del lado cliente. Estos controles se alojan del lado del framework. Veamos en la **Tabla 5** una descripción de los controles que nos permitirán crear las **UI** de nuestras aplicaciones basadas en la Web.

TIPOS DE CONTROLES	DESCRIPCIÓN
Controles HTML Server	Son un equivalente a los controles HTML estándar, pero corresponden a la estructura del framework .NET.
Controles de Web Form	Son controles nativos propios de ASP.NET. En su mayoría, duplican y mejoran la funcionalidad de los controles de servidor HTML.
Controles de validación	Son útiles para validar el contenido de los campos de texto, antes de procesar un Form o guardar la información en una base de datos.
Controles de lista	Aquí se incluyen los controles ASP.NET del tipo Cuadro de lista, DropDownList, ListBox, CheckBoxList y RadioButtonList.
Controles de plantilla	DataList, DataGrid y Repeater son controles personalizables en cuanto a aspecto gráfico y comportamiento mediante el uso de plantillas.
Extensiones AJAX	Estos controles de servidor son funcionalidades JavaScript encapsuladas para refrescar contenido en páginas sin necesidad de recargarlas.
Otros controles	Calendar, AdRotator y XML son controles que no entran en ninguna de las categorías mencionadas anteriormente.

***Tabla 5.** Controles ASP.NET y sus funciones principales.*

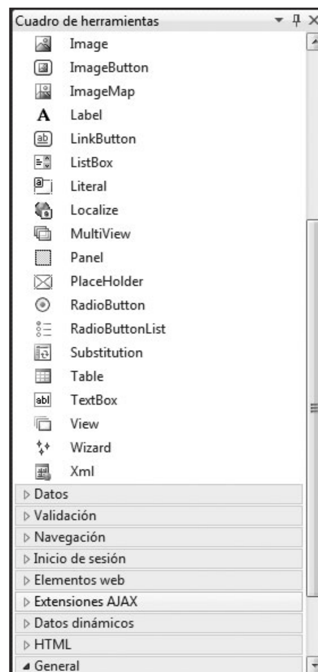


***Figura 17.** Cuadro de herramientas de controles ASP.NET.*

Controles HTML Server

Los controles HTML Server se encuentran definidos en el **namespace System.Web.UI.HtmlControls**. Son los más comunes para crear UI. Fueron inspirados en una mezcla de los controles comunes de aplicaciones **Windows Forms** y los **controles HTML**, con la ventaja de poder mejorar sus prestaciones comparándolos con los controles clásicos que se pueden crear desde cualquier aplicación de diseño web. Iniciemos un nuevo proyecto para agregar algunos controles y estudiarlos un poco más de cerca en cuanto a propiedades y métodos. El proyecto se llamará **ControlesWebForms**. Editamos en modo **Diseño** el archivo **Default.aspx** y eliminamos el texto que contiene su página central. En el primer ejemplo, agregamos un **textBox**, un **Label** y un control Button. En esta oportunidad, vamos a agregar un control **UploadFile**, un control **Button** y un control **Label1**. El control **Button** tendrá como valor **Text = Subir Archivo**. La distribución de los controles en pantalla será la de la **Figura 18**.



***Figura 18.** Ubicación de los controles en la página.*

Luego, hacemos doble clic en el control **Button1** y agregamos el siguiente código en el evento **Click()** de este control:

```
If (FileUpload1.HasFile) Then
    SubirArchivo(FileUpload1.PostedFile)
Else
    Label1.Text = “No se especificó un archivo a subir.”
    Exit Sub
End If
```

A través de este código verificamos si existen un archivo a subir. De ser cierto, entonces ejecutamos el procedimiento que lo sube a nuestra web, sino, notificamos al usuario mediante un mensaje en pantalla.
Creemos ahora un procedimiento SubirArchivo() luego del inicio de la clase:

```
Public Class _Default
     Inherits System.Web.UI.Page
     Sub SubirArchivo(ByVal file As HttpPostedFile)
          Dim Ruta As String = "c:\temp\prueba\"
          Dim archivo As String = FileUpload1.FileName
          Dim rutaCompleta As String = Ruta + archivo

          If (System.IO.File.Exists(rutaCompleta)) Then
               Label1.Text = "Existe el archivo que quiere subir."
               Exit Sub
          Else
               Label1.Text = "Se ha subido el archivo seleccionado."
          End If
          FileUpload1.SaveAs(rutaCompleta)
     End Sub
...
```

FileUpload1.HasFile nos devuelve un valor **booleano** para saber si se seleccionó un archivo o no. Luego, declaramos el procedimiento **SubirArchivo**, pasándole como parámetro el archivo que elegimos para subir al servidor. El tipo de variable que le pasaremos debe ser **HttpPostedFile**, con lo cual .NET interpreta que estamos subiendo un archivo. Creamos la siguiente ruta en el disco de nuestra computadora: **C:\temp\prueba**. Dentro del procedimiento **SubirArchivo(...)** estamos indicando que la ruta donde se alojará es la declarada en la variable **Ruta**. La variable **rutaCompleta** es la encargada de concatenar el nombre de archivo con la ruta donde se guardará el mismo, y la sentencia **FileUpload1.SaveAs** queda como la responsable de guardar el archivo seleccionado. Para el caso de que todo vaya correcto, **Label1** nos avisará esto mediante la leyenda correspondiente.
Si existe el archivo, **Label1** nos notificará de ello, y si no seleccionamos ninguno, también nos notificará. Ejecutemos el proyecto y veamos cómo funciona. En la Figura 19 podemos ver el resultado de este proceso.



CONTROLES CLÁSICOS HTML

Los controles clásicos HTML son los mismos que se pueden crear con Dreamweaver, por ejemplo, o con otro diseñador de páginas web. Son limitados en comparación con los incluidos en ASP.NET, con lo cual se desaconseja su uso. En caso de aplicarlos, deberemos reemplazar varias funciones directamente con JavaScript, lo que ralentizará el sistema.



***Figura 19.** Podemos ver el mensaje devuelto por la página luego de seleccionar un archivo. Este archivo fue subido con éxito a la carpeta, como observamos en el lateral izquierdo de la imagen.*

Para mejorar el detalle de la pantalla, cambiemos la línea **Label1.Text = "Se ha subido el archivo seleccionado."** por lo siguiente:

```
Label1.Text = "Se ha subido el archivo " + "<b>" + archivo + "</b>"
```

Podemos aprovechar sentencias propias de HTML para darles vida a los textos de los **Labels**, como resaltándolos, aplicándoles color u otro tipo de formatos.
El control **Image** funciona casi igual que en los **Windows Forms**. En lugar de tener una propiedad **Picture** o propiedad **Image**, en este caso tiene una propiedad llamada **ImageURL**, donde se puede especificar el **Path** del archivo de imagen por mostrar. Ya habiendo conocido un poco mejor las diferencias entre **Image** de WindowsForm y el control equivalente para ASP.NEt, probemos el mismo para aprender cómo funciona.
Agreguemos un control Image, y ajustemos las propiedades **Width = 227px** y **Height = 60p**, luego agreguemos en **Form_Load()** el siguiente código:

```
Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        Image1.ImageUrl = http://www.redusers.com/noticias/wp-
content/themes/ru50/images/redusers-header.png
...
```

Ejecutemos la página y veremos cómo se carga una imagen desde un Parh completo. Es posible elegir otra imagen si lo deseamos. Veamos a continuación otros controles más, pero antes, para no sobrecargar una página web, solo editaremos Site.Master y crearemos otras páginas para trabajar más cómodamente. Abrimos el archivo **Site.Master** y seleccionamos el Menú Principal superior, que consta de dos botones: **Página Principal** y **Acerca De**. Hacemos clic sobre la flecha que aparece sobre el lado derecho y nos encontraremos con **Tareas de Menú**, desde donde podemos agregar o quitar los botones existentes y configurar también las páginas a las que ellos nos redirigen. Seleccionamos la opción **Editor de Elementos del menú**. Renombremos Página Principal como Subir Archivos, tanto en su propiedad **Text** como en su propiedad **Value**. Agregamos a continuación un menú, desde el botón **Agregar un elemento de Raíz**. En la propiedad **Text** escribimos Otros Controles, y en **NavigateURL**, **~/OC.aspx**. Repetimos la operación añadiendo un nuevo elemento de menú, en su propiedad **Text** escribimos **Validación y lista**, y en la propiedad **NavigateURL**, **~/Validacion.aspx**.

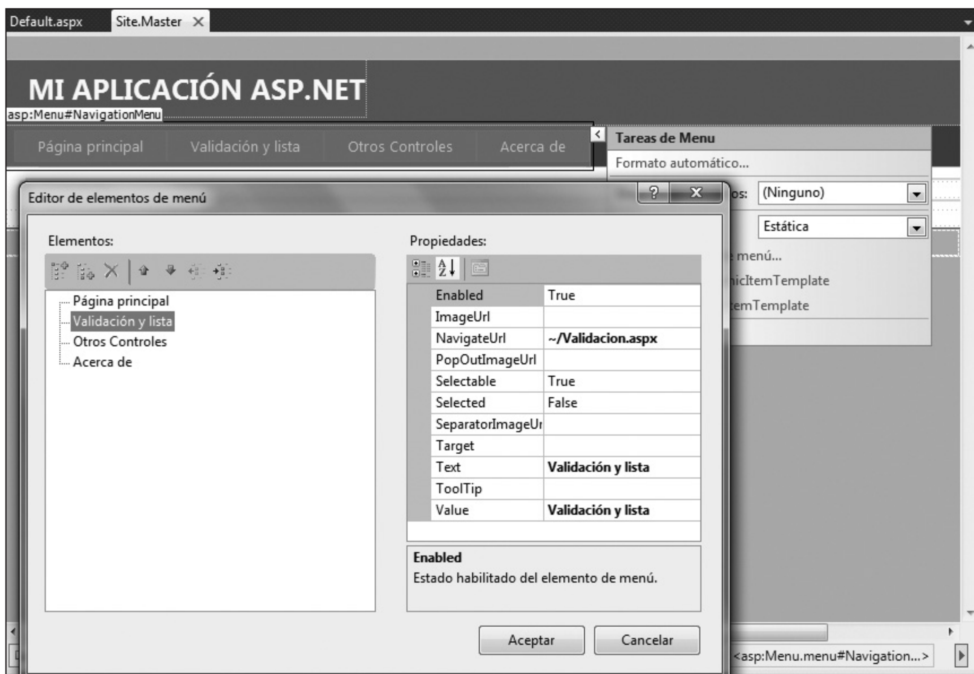


Figura 20. *Editor de menú de Site.Master.*

Agregamos, a continuación, un nuevo Web Form presionando **CTRL + SHIFT + A**; seleccionamos **Web Forms que utilizan una página maestra**, lo guardamos con el nombre **OC.aspx** y, luego, elegimos **Site.Master** como página maestra. Repetimos lo mismo y agregamos un nuevo **Web Forms**, al que guardamos con el nombre **Validacion.aspx**.

Controles de validación y lista

Diseñaremos a continuación un **Web Form** con algunos campos simulando ser un formulario para envío de información, y utilizaremos algunos **controles de validación** para verificar que los datos ingresados sean correctos. En el Web Form **Validacion.aspx** agregamos algunos controles **Label** y **TextBox**, y un control **Button**.

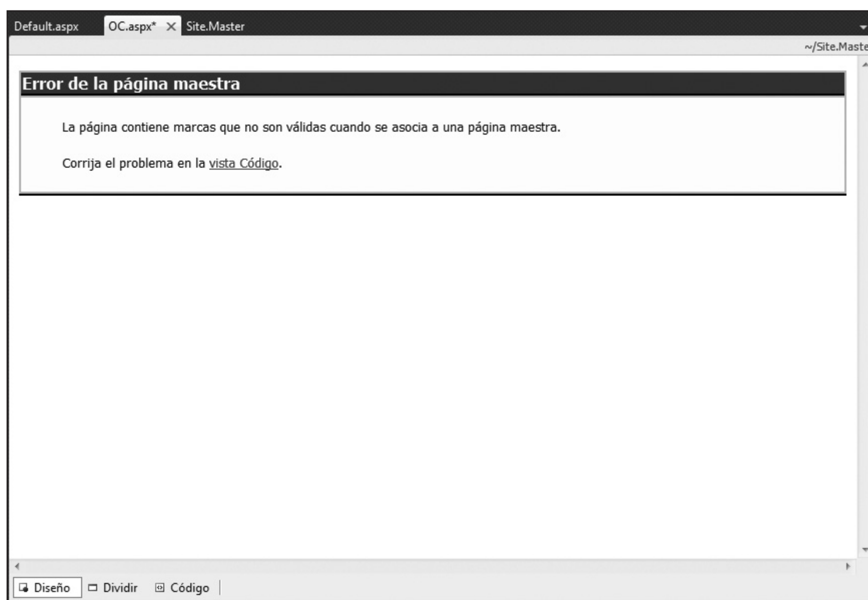


***Figura 21**. Error típico de los Web Forms vinculados a un Site.Master que, a su vez, contienen código HTML.*

Ajustamos la propiedad **Text** de cada uno de los **Label control** con el siguiente texto: **Nombre**, **Apellido**, **Edad** e **Email**. Luego, al lado de cada uno de ellos situamos los controles **TextBox**. Al final de Form añadimos un control **Button**, cuyo nombre será **btnValidar** y su propiedad **Text** será **Validar**. De esta manera, ya tenemos un formulario de contacto simulado. La finalidad de este formulario de contacto no será enviarlo por correro o guardar la información en una base de datos, sino validar el contenido de cada uno de sus campos. Los dos primeros campos son del tipo texto,



CONTROLES ASP.NET AJAX

Luego del lanzamiento de Visual Studio 2005, Microsoft lanzó al mercado una suite de controles ASP.NET AJAX que permiten actualizar y refrescar datos en pantalla sin necesidad de recargar la página. Visual Studio 2008 y Visual Studio 2010 ya los incluyeron por defecto, a diferencia de VS2005, en que deben descargarse e instalarse por separado.

el tercero es del tipo numérico, y el cuarto es una dirección de correo electrónico. Validar a mano cada tipo de dato que se guardará en un campo es una tarea engorrosa, que volverá lento nuestro sitio web. Además, casi siempre sucede que las rutinas que deben validar tipos de datos terminan por contener uno o más **bugs**. Desde la versión 2003 de .NET se incluyeron los **controles de validación y lista**, que, a través de una serie de parámetros, pueden validar de manera directa y sin pérdida de tiempo el tipo de dato cargado en los campos de nuestros **Web Forms**.



Figura 22. *Vista de los controles de validación en el cuadro de herramientas y cómo quedan en pantalla en la página web.*

Para los dos primeros campos, agregamos un control **RequiredFieldValidator** por cada uno de ellos. Para el campo **Edad** agregamos un control **RangeValidator**, y para **Email**, un control **RegularExpressionValidator**, que obliga al usuario a completar el campo al cual está vinculado. **RangeValidator** nos servirá para especificar un rango determinado, para este caso, numérico, que podrá ingresarse dentro del campo **edad**. Por último, el campo **Email** lleva formato de correo electrónico, con lo cual puede variar dependiendo de cada nación. Veamos,



BUG O ERROR DE SOFTWARE

Se conoce como **bug** a aquellos errores de software que surgen durante el proceso de creación o ejecución de un programa de computadoras. Esto sucede en cualquier etapa del ciclo de vida del soft, aunque por lo general se da en la etapa de desarrollo. Su nombre data de 1947, cuando los creadores de **Mark II** encontraron un cortocircuito causado por una polilla.

entonces, cómo ajustar cada campo. **RequiredFieldValidator** tiene una propiedad llamada **ControlToValidate** en la cual especificamos el **textBox** que queremos validar. La propiedad **Text** nos permite ingresar el mensaje que aparecerá en pantalla cuando este campo no cumpla con la validación correcta. Ajustamos los campos correspondientes para que valide que cada uno de ellos contenga un texto ingresado. El control **RangeValidator** posee la propiedad **ControlToValidate**, donde le indicamos qué campo deberá validar. En la propiedad **Type** le decimos que solo podrá permitir números enteros, dado que se tratará de una edad; y las propiedades **MinimumValue** y **MaximumValue** son las encargadas de validar que en dicho campo se cumpla con el **rango etario**. Al igual que **RangeValidator**, la propiedad **Text** contendrá el mensaje de error que aparecerá si no se cumple con la condición. Por último, nos queda **RegularExpressionValidator**, que tiene las mismas propiedades **ControlToValidate** y **Text**, e incluye una más, que es **ValidationExpression**, en la cual podemos especificar, a través de la herramienta **Editor de expresiones regulares**, qué tipo de campo deberá validar.

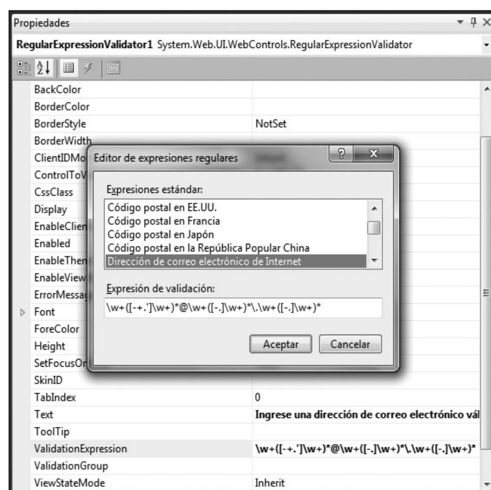


***Figura 23.** En esta figura podemos ver las diferentes opciones del Editor de expresiones regulares.*

Agregamos los mensajes correspondientes en las propiedades **Text** y ajustamos cada uno de los controles por validar. Para **RequiredFieldValidator1** pondremos que debe validar el contenido de **TextBox1**. **RequiredFieldValidator2** hará lo propio con **TextBox2**. **RangeValidator1** validará que las edades ingresadas en **TextBox3** estén entre 18 y 65 años, y por último, queda el campo **Correo electrónico**, que validará que la dirección ingresada en **TextBox4** tenga la composición de un correo electrónico. Agregamos cada mensaje en cada uno de los validadores y presionamos la tecla **F5**. Provocamos los errores para recibir los mensajes pertinentes y presionamos el botón **Validar**.

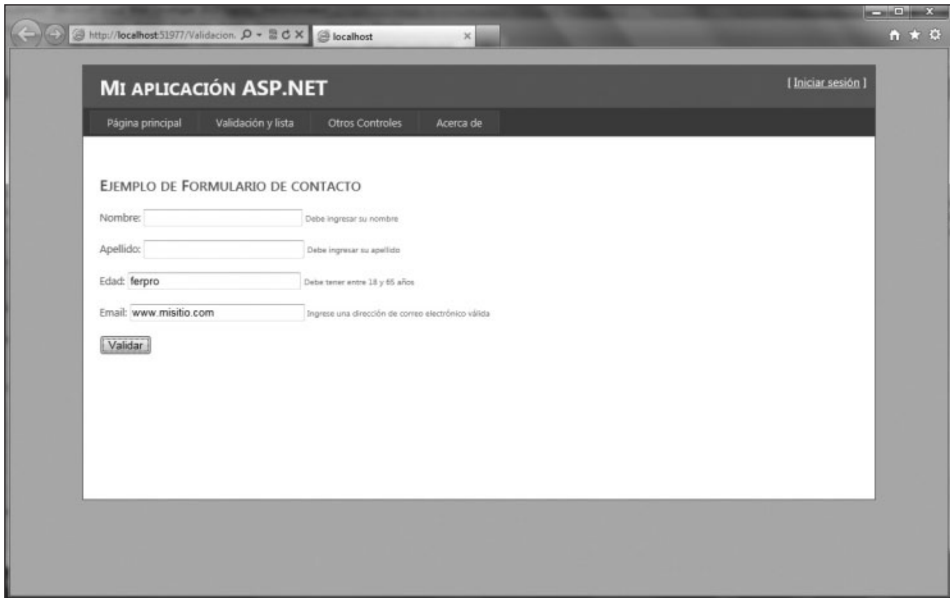


Figura 24. *Los controles de validación funcionando correctamente.*

Los controles de lista son controles que heredan su base principal de la clase **ListControl**, con lo cual tienen muchas propiedades, métodos y eventos en común, además de los heredados de la clase **WebControl**. Los controles son **DropDownList**, **ListBox**, **RadioButtonList** y **CheckBoxList**. En la **Tabla 6** podemos ver las propiedades comunes a todos estos controles:

SINTAXTIS	DESCRIPCIÓN
Items	Colección de objetos ListItem hijos.
SelectedIndex	El índice del elemento que fue seleccionado (es de lectura y escritura).
SelectedItem	Indica el objeto ListItem seleccionado (sólo lectura).
AutoPostBack	Cuando es True, el control devuelve los datos cuando cambia el valor SelectedIndex.

Tabla 6. *Estos son los principales miembros de la clase* ***ListControl****, que se comparten entre los controles de lista mencionados.*



XML-BASED USER INTERFACE LANGUAGE

XUL es el término que define a una aplicación XML basada en la interfaz de usuario. La ventaja de esto es que nos aporta una definición de GUIs simples y portables, lo que reduce el trabajo en el desarrollo de software. Mozilla es el principal implementador de este lenguaje, combinando el desarrollo de interfaces XUL con JavaScript.

Abrimos la página **OC.aspx** y agregamos los controles aquí mencionados para comprender un poco mejor su manera de actuar. El primer control por agregar en el Web Form es **CheckBoxList**. Podemos ver en él, y también en los próximos por agregar, que disponemos de una opción de **Tareas**. Si seleccionamos **Editar elementos**, podemos agregar de manera estática aquellos que lo compondrán. Vamos a añadir algunos ítem, tal como muestra la **Figura 25.**

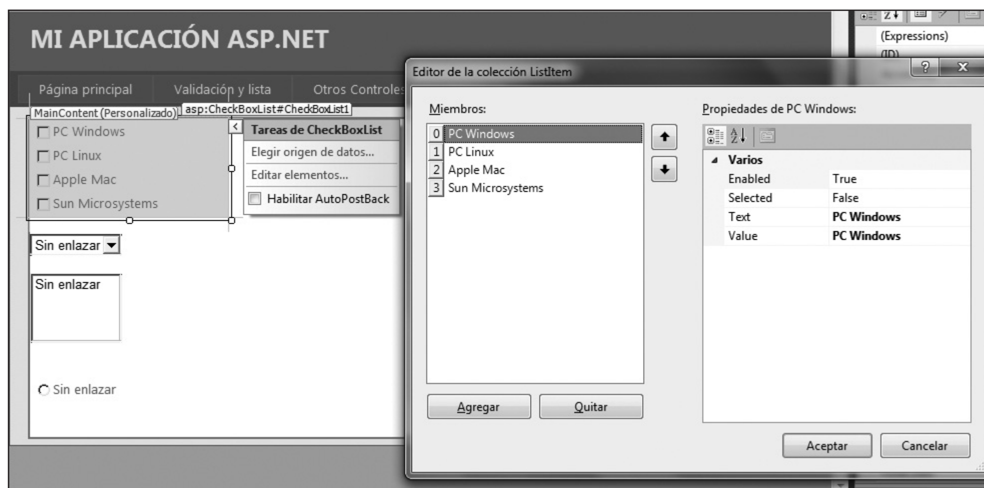


***Figura 25.** El editor de la colección ListItem permite agregar o quitar elementos estáticos a nuestras páginas web. En el lateral izquierdo de cada ítem vemos el **ID** asignado a cada uno de ellos.*

La opción **Elegir origen de datos** de las **Tareas de CheckBoxList** permitirá enlazar al contenido de una base de datos para que, en vez de estático, dicho control sea dinámico, basándose en la totalidad de los registros de la **tabla** o **consulta** de donde tomemos los datos. Esto mismo vale para el resto de los controles de lista. Tal vez algunos conceptos confundan un poco. Usarlos nos aclarará mejor el panorama. Agreguemos el resto de los controles mencionados al Form, y procedamos a rellenarlos de la misma manera.

Otros controles web

Entre la importante variedad de controles web que podemos agregar a nuestros desarrollos nos queda por ver **Calendar**, **AdRotator** y los controles de Inicio de sesión. **Calendar** tiene un sinnúmero de propiedades y sucesos, por lo cual, si deseáramos conocer en detalle cada uno, deberíamos dedicarle un capítulo completo, porque es uno de los controles más complejos que posee ASP.NET, después de **DataGrid**. Este control sorprende a la mayoría, dado que permite personalizar al máximo el aspecto de cualquier elemento visual que tiene. También permite seleccionar de manera individual días, semanas o meses, agregar texto o imágenes dentro de cada celda de día

individual y controlar si se puede seleccionar, y un día o más. El control **Calendar** posee las categorías **Style** y **Appearence**, que nos permitirán configurar en detalle todos sus objetos.

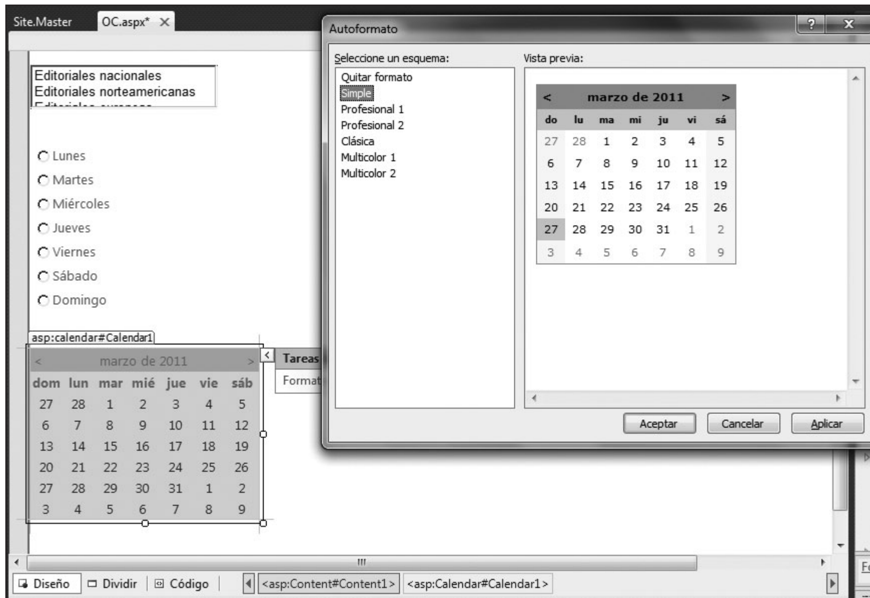


Figura 26. *El control Calendar incluye una serie de formatos predeterminados que nos evitarán trabajar al máximo en personalizar en detalle su estilo.*

El principal objeto del control **Calendar** es **Day**, que describe el día que ha sido seleccionado. También dispone de propiedades como **Date**, **DayNumberText**, **IsToday**, **IsWeekend**, **IsOtherMonth**, **IsSelected** e **IsSelectable**. Todas son de solo lectura, pero lo suficientemente útiles para manejar en detalle el control **Calendar**. El control **AdRotator**, al igual que su componente del mismo nombre que se podía encontrar en ASP, tiene la capacidad para mostrar avisos de manera aleatoria. La diferencia entre el viejo **AdRotator** y este es que el último levanta toda la información de un archivo con formato XML. El formato del archivo Extended Markup Language sería similar al siguiente ejemplo en código:

LA CLASE STYLE

Esta clase encapsula las propiedades que manejan la apariencia de un control de servidor web. Con ellas se pueden aplicar a distintos controles de servidor, lo que permitirá tener una apariencia unificada en todo el website. Desde **BackColor**, **ForeColor**, **Font**, **FontColor** y otras tantas propiedades más, podemos llevar la personalización de un control web al máximo detalle.

```
<? xml versión="1.0" encoding="utf-8"  ?>
<Advertisements
Xmllns=http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File>
<Ad>
<ImageUrl>/ru_banner.gif </ImageUrl>
<NavigateUrl>www.redusers.com</NavigateUrl>
<AlternateText>El mejor portal de tecnología. </AlternateText>
<Keyword>Noticias, Tecnología</Keyword>
<Ad>
<ImageUrl>/ru_banner.gif </ImageUrl>
<NavigateUrl>www.usershop.com.ar</NavigateUrl>
<AlternateText>El mejor portal de revistas y libros de tecnología.
</AlternateText>
<Keyword>Libros, Revistas, Fascículos, Tecnología</Keyword>
</Ad>
…
</Advertisements>
```

Los controles de **Inicio de sesión** son un conjunto de herramientas prearmadas que nos ofrece Microsoft para armar un sitio que requiera usuario y contraseña para acceder. A través de este conjunto de herramientas, tenemos todos los controles prearmados para generar el alta de un nuevo usuario, el cambio de contraseña, el inicio de sesión, el envío de la clave en base a una dirección de correo electrónico y algunas opciones óptimas más que nos harán ahorrar mucho tiempo. El tipo de validación queda a cargo del programador, quien puede validar contra una **base de datos**, y guardar el nombre de usuario en una **cookie**, un **objeto Session**, o incluso, validarlo contra el nombre de usuario y contraseña de dominio de un Windows Server, dado que el servidor web que alojará nuestro site estará instalado sobre un servidor Microsoft. Dentro del cuadro de herramientas, encontramos una pestaña llamada **Inicio de sesión**, en la que se incluye el conjunto de controles **ChangePassword**.



FACILITAR AL MÁXIMO EL DESARROLLO WEB

Microsoft buscó que Visual Studio incluya la mayoría de los controles o conjuntos de controles que faciliten y reduzcan los tiempos de desarrollo web. Gracias a esto, con los controles de inicio de sesión podemos desarrollar aplicaciones que permitan registrarse, loguearse a nuestro sistema y recuperar las claves de usuario con tan solo unas pocas líneas de código, en muy poco tiempo.

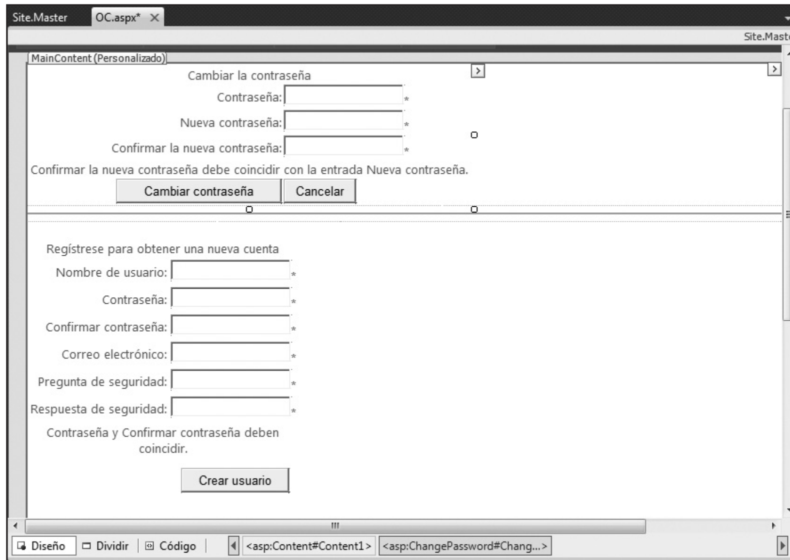


Figura 27. *Controles prearmados para cambio de contraseña.*

Las tareas de este control permiten especificar el look entre varios tipos de **Formatos predeterminados**, o establecer nuestro propio formato. Posee dos vistas diferentes: **Cambiar contraseña** y **Correcto**, siendo este último el paso final cuando uno ya cambió la contraseña. Otro control disponible es **CreateUserWizard**, que nos brinda un conjunto de campos determinados para crear un nuevo usuario.
Dichos campos pueden ser modificados tan solo agregando los que necesitemos y quitando los que no hacen falta.

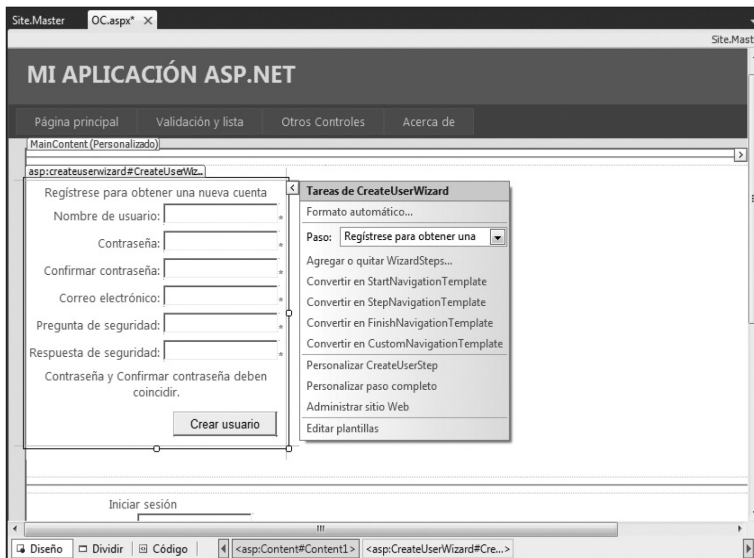


Figura 28. *Conjunto de controles para la creación de un usuario.*

Al igual que **ChangePassword**, se puede establecer su look entre distintos formatos predeterminados. En sus tareas es posible editar y modificar los distintos pasos necesarios para crear un nuevo usuario, tal como los **Asistentes** que solemos encontrar en los sitios web de Microsoft. **Login** es otro conjunto de controles que nos permiten crear una ventana de inicio de sesión para nuestro sitio web. También posee la capacidad de establecer su aspecto desde su menú de **Tareas**. En el cuadro de propiedades de **Inicio de sesión**, es posible agregar opciones dentro del conjunto de control para, por ejemplo, enviar el usuario a una **URL específica**, en la cual puede haber una página web con el conjunto de controles **CreateUserWizard**, para que se cree un usuario en caso de que no lo tenga. También se puede especificar la **página de destino** una vez logueado el usuario, la **página de error**, una **página de ayuda**, un link para la **recuperación de la contraseña**, y un sinfín de opciones más.

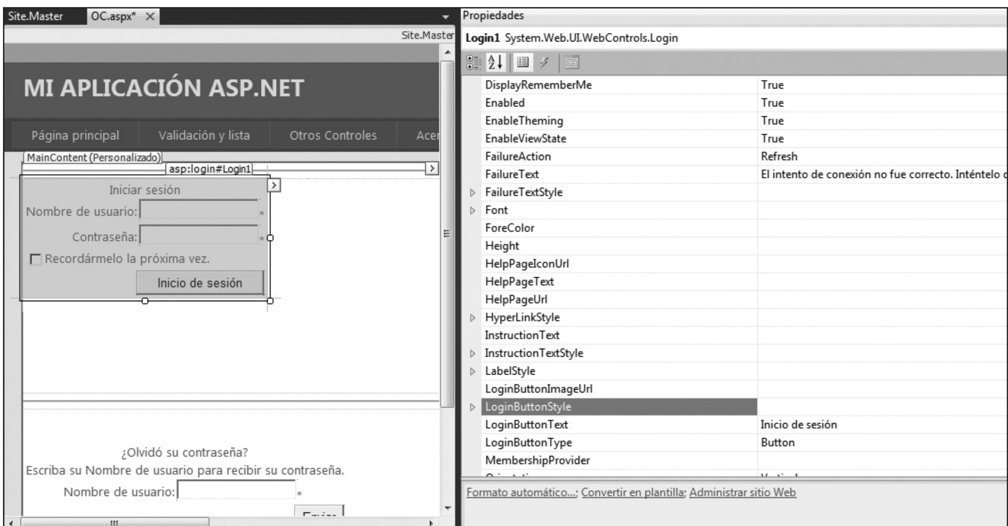


***Figura 29.** Conjunto de controles de login y sus propiedades.*

Para terminar, **PasswordRecovery** es el conjunto de controles que nos permite recuperar la contraseña en caso de extravío u olvido. Estos controles tienen el estado principal, que es **ingresar el nombre de usuario**, un segundo estado que es la

BASES DE DATOS E ISP

Cuando se desarrolla una web ASP.NET, nos encontramos con el dilema de cómo declarar el nombre del servidor de base de datos dentro de la conexión ASP.NET. En casi todos los proveedores, esto se hace mediante **localhost** o, en su defecto, la dirección IP **127.0.0.1**. Igualmente, contamos con la ayuda del soporte técnico del hosting.

Pregunta de recuperación y un tercer estado que aparece solo **si la respuesta fue correcta**. Al igual que los controles anteriores, también tiene una opción de **Tareas**, desde donde se puede configurar el aspecto, y un **Cuadro de propiedades** con todas las opciones más comunes que encontramos en cualquier sitio web que brinda el servicio de recuperación de contraseñas para usuarios.

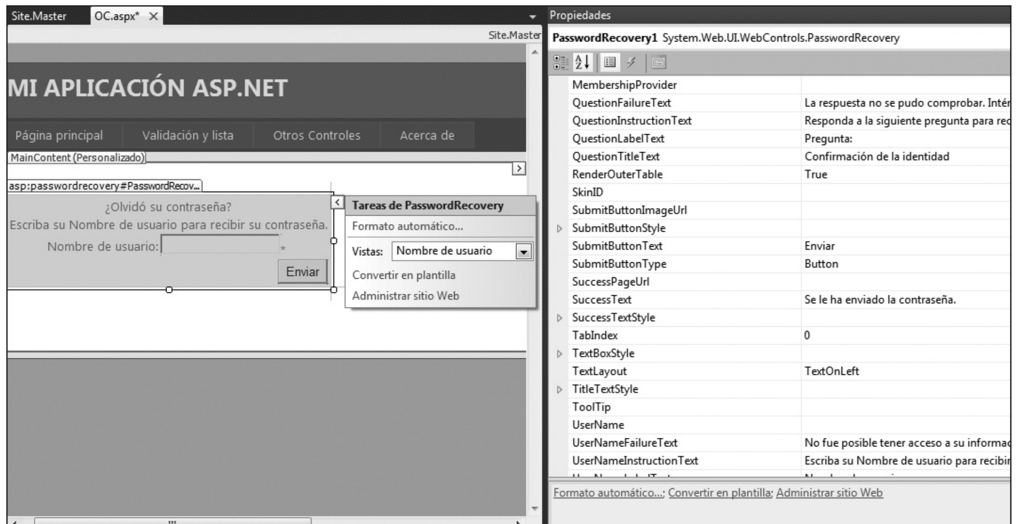


Figura 30. *Conjunto de controles para la recuperación de la contraseña.*

Cómo enlazar datos con Web Forms

Al igual que las aplicaciones Windows Forms, en ASP.NET también podemos desarrollar sitios web **enlazados a datos**. Vamos a iniciar un nuevo proyecto del tipo **Aplicación Web ASP.NET**. En la página principal, **Default.aspx**, agregamos un control **GridView** desde la pestaña **Datos** de **Cuadro de herramientas**. En las **Tareas de GridView** seleccionamos **Formato Automático** y desde allí, algún estilo de grilla que nos guste entre todas las opciones listadas. Luego elegimos **Nuevo Origen de datos** para establecer la conexión a una base de datos. Entre las opciones ofrecidas por el Asistente, seleccionamos **Base de datos**. Luego elegimos **Nueva conexión** y, como **Origen de datos**, **Archivo de base**



ESTÁNDAR ATOM

ATOM es un formato de redifusión de información basado en archivos XML. Permite redifundir información web a través del nombrado protocolo ATOM. Este protocolo está basado en HTTP para crear o actualizar recursos de la Web. Las fuentes permiten que determinadas aplicaciones busquen actualizaciones de contenido web, similar al usado mediante RSS.

de datos Sql Server. Presionamos el botón **Continuar**, y desde la ventana **Agregar conexión**, seleccionemos el botón **Examinar** para buscar el archivo de datos. Buscamos en el disco de nuestra computadora si disponemos de la base de datos **Northwind.mdf**. En caso de que esté allí, copiemos la ruta para seleccionarla. Si no llegamos a disponer del archivo, podemos descargarlo desde el siguiente link: **www.microsoft.com/Downloads/details.aspx?FamilyID=06616212-0356-46a0-8da2-eebc53a68034&displaylang=en**. Elegimos esta base de datos, presionamos **Probar conexión**, y si esta fue satisfactoria, presionamos **Aceptar** para finalizar. Avanzamos al siguiente paso hasta encontrar las tablas listadas en la base. Seleccionemos la tabla **Contacts** y, de ella, los siguientes campos: **ContactID**, **ContactType**, **CompanyName**, **ContactName** y **ContactTitle**. Si presionamos **Consulta de prueba**, tendremos una vista previa de los registros. Presionamos el botón **Finalizar**, y se terminará de armar el enlace de datos con la tabla de la base de datos. Al igual que con el control **DataGridView** de **Windows Forms**, desde la opción **Tareas** de **GridView/Editar columnas**, podemos darles forma, elegir el tipo de fuente y su tamaño, y el ancho de cada columna de la grilla de datos.

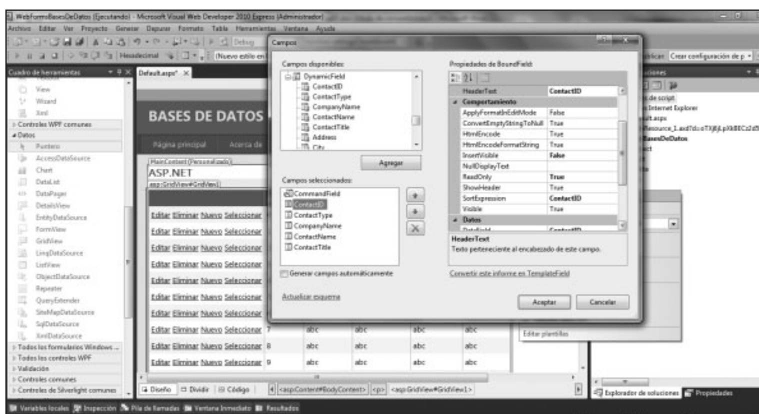


***Figura 31.** Edición de los campos de GridView.*

Si prestamos atención, notaremos que el primer campo creado en la grilla no es de la tabla de datos, sino del tipo **CommandField**. Este tipo de celda genera los links necesarios para seleccionar registros, y desde la tabla de propiedades **Edición de campos**, podemos agregar otras funcionalidades más, para manejar un **ABM** completo a través de links. Seleccionamos el campo **CommandField** y, en sus propiedades, ponemos con valor **True** a **ShowDeleteButton**, **ShowEditButton**, **ShowHeader**, **ShowInsertButton** y **ShowSelectButton**. Si deseamos cambiar los textos mostrados para estas acciones sobre los registros, podemos modificar las propiedades **UpdateText**, **SelectText**, **NewText**, **InsertText**, **EditText**, **DeleteText** y **CancelText**. Y desde la propiedad **ButtonType** podemos especificar el tipo de **commandField** deseado entre **Button**, **Text** e **Image**.

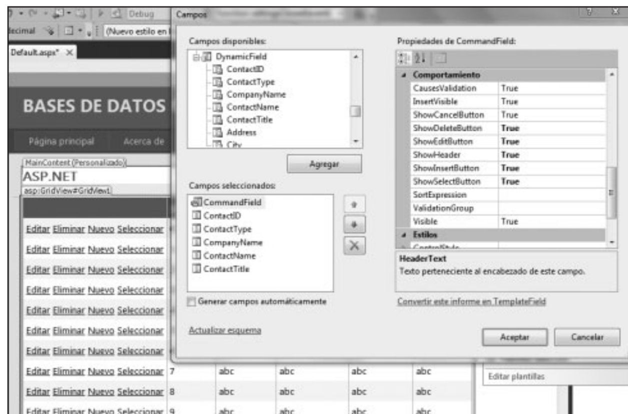


Figura 32. *Inclusión de campos* ***CommandType*** *en* ***GridView****.*

Ejecutamos entonces nuestro proyecto de base de datos y vemos el resultado de la personalización del control **GridView**.

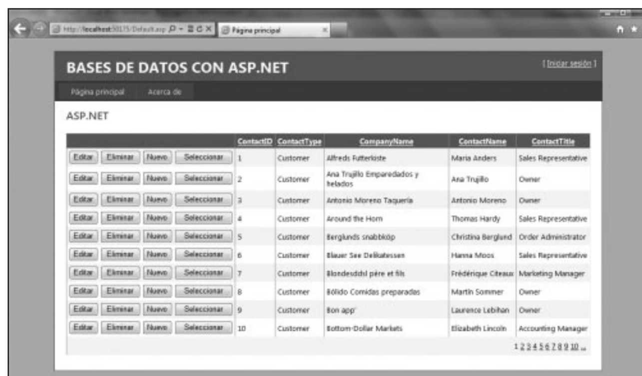


Figura 33. *Un ABM de contactos realizado con algunos pasos y sin código de por medio, gracias al Asistente de conexión a fuentes de datos que incluye VB 2010.*

RESUMEN

Dentro del capítulo ASP.NET hemos visto un pantallazo de cómo generar aplicaciones web con Visual Basic 2010. Lo que ofrece ASP.NET es facilitar al máximo el trabajo del desarrollador, y acercarlo al diseño web, con la posibilidad de personalizar cualquier desarrollo con hojas de estilo CSS que permitirán mantener una línea de trabajo clara y estéticamente prolija, al igual que lo hace un diseñador. También vimos cómo, con pocos clics del mouse y en un tiempo casi récord, podemos armar un ABM de datos conectándonos a una base SQL Server. Los conocimientos hasta aquí obtenidos nos dan las herramientas necesarias para seguir explotando el poder de ASP.NET sobre la Web.



ACTIVIDADES

PREGUNTAS TEÓRICAS

1. Investigue qué diferencias hay entre Microsoft Web Developer Studio 2010 y Microsoft Expression Blend 4.
2. Investigue las diferencias entre un sitio HTTP y un sitio HTTPS.
3. ¿Con qué requisitos debe contar un website para utilizar HTTPS?
4. ¿Sobre qué tipo de computadora debe montar un servidor web para que sus aplicaciones ASP.Net funcionen?
5. ¿Cuál es el componente ASP.NET que debe utilizar para subir archivos a un sitio web creado mediante ASP.NET?
6. Este mismo componente, ¿le permite también realizar descargas de archivos al disco de una computadora?
7. Investigue qué es LINQ y para qué se utiliza.
8. ¿AJAX.NET viene incluido en el paquete de desarrollo de aplicaciones ASP.NET, o debe descargarse aparte?
9. Investigue el uso básico de AJAX.NET, luego intente crear una aplicación simple donde pueda aplicar esta tecnología.
10. ¿ASP.NET permite utilizar un solo lenguaje de desarrollo en una aplicación web, o puede combinar Visual Basic 2010 y Visual C# dentro de un mismo sitio web?

XAML y WPF

En este capítulo veremos las alternativas propuestas por Microsoft para desarrollar aplicaciones del tipo Rich User Interfaces desde el IDE de Visual Basic 2010, para que corran tanto en Windows como en un navegador web. Para esto, haremos un repaso por el lenguaje de marcado Extensible Application Markup Language y por Windows Presentation Foundation, conocidos como XAML y WPF. También llevaremos a cabo algunos ejemplos prácticos para comprender mejor el desarrollo sobre esta plataforma.

QUÉ ES XAML

Desde la aparición de Internet, mucho se ha modificado la forma de trabajar y desarrollar software. Los clásicos programas instalables en Windows debieron cambiar para poder ser útiles desde la Web. Internet también trajo un sinfín de herramientas y soluciones prácticas para los usuarios, y junto con ellas, se fueron descubriendo nuevas maneras de desarrollar aplicaciones. **XAML** es una variante para el desarrollo de aplicaciones conocidas como **RIA** (*Rich Interface Application*). Sus siglas provienen de *eXtensible Application Markup Language*, y su pronunciación en inglés es "**Zammel**". Entre las primeras herramientas que permitieron crear este tipo de aplicaciones se encontraban **Adobe Dreamweaver** y **Photoshop**. Desde ellas se podía realizar el front end de la aplicación en formato **XML** para programas que corrieran sobre Windows, y en formato **xHTML** para los que lo hicieran sobre la Web. Este nuevo lenguaje XAML define los elementos por utilizar en la interfaz de usuario, separándolos del código fuente, que es el que hace que dicha aplicación funcione. Todo el contenido visual con el que se trabaja dentro de una aplicación XAML es tratado como objeto, cuyo elemento raíz es la clase **Window**. Los controles definidos dentro de XAML hacen referencia a las clases del framework .NET. El lenguaje de marcado es un lenguaje propio derivado de XML y no debemos confundirlo con el lenguaje de programación utilizado en Visual Basic 2010.

Uno de los beneficios de XAML es su capacidad de separar una aplicación en dos capas. Por un lado, está la interfaz visual, y por otro, el código fuente que le da vida a nuestro desarrollo. Esto facilita que en la creación de un sistema, en caso de que debamos cambiar algo estético o simplemente visual, podamos hacerlo sin tener que recompilar nuestra app. La UI o User Interface se almacena en un archivo con extensión **.XAML**, con lo cual este archivo puede ser desarrollado por un diseñador y, luego, es posible importarlo a nuestro proyecto Visual Basic 2010 para escribir el código que le dé vida. Veamos a continuación un poco de sintaxis XAML para comprender mejor la teoría hasta aquí explicada:

```
<Window x:Class="MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Ventana XAML" Height="480" Width="640">
</Window>
```

En este ejemplo nos encontramos con etiquetas iniciadas por el carácter **<** y cerradas por el carácter opuesto, **>**. Dentro de ellas se escribe el código, y se finaliza con la misma etiqueta inicial antecedida de una barra **/**.

Windows Presentation Foundation

WPF es el nombre con el que Microsoft bautizó a la tecnología que permite el desarrollo de aplicaciones XAML. En un futuro, se pretende reemplazar el desarrollo de aplicaciones de escritorio directamente por aplicaciones WPF. Estas últimas aprovechan más los **recursos gráficos** que se incluyeron en la plataforma Windows, a partir de **Windows Vista**. Para comprender mejor esto, iniciemos un nuevo proyecto en Visual Basic 2010. En la ventana **Nuevo Proyecto**, vamos a elegir, entre las plantillas disponibles, la opción **Aplicación WPF** y le damos como nombre **EjemploWPF**.

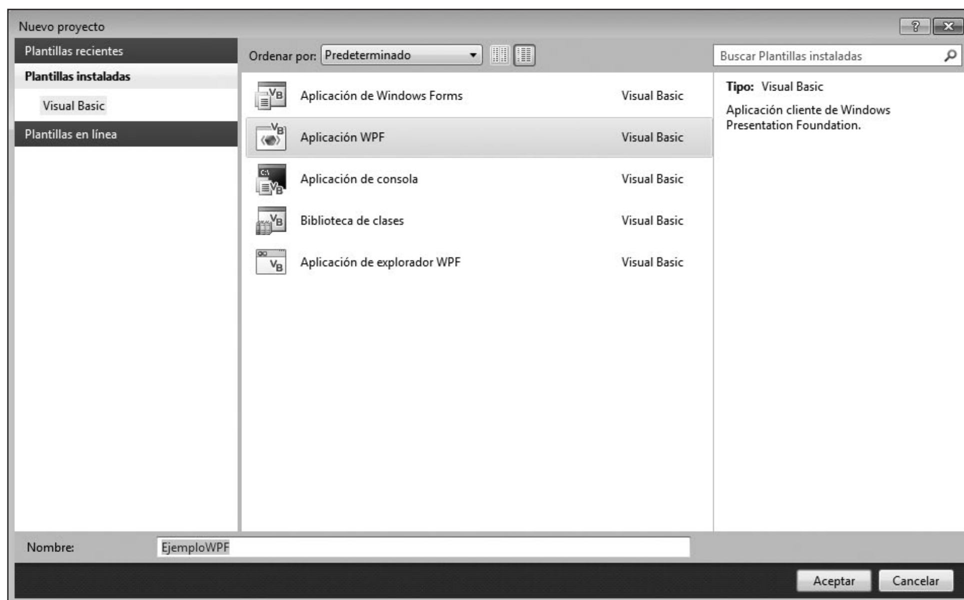


***Figura 1.** Comencemos el desarrollo de nuestra primera **Aplicación WPF** o **XAML** para conocer su IDE.*

El IDE es similar a las aplicaciones **Windows Form** que vimos en los primeros capítulos de este libro. La pestaña principal contiene un **WindowForm** denominado **MainWindow**, que grabará todo el diseño visual del Form en el archivo **MainWindow.xaml**; hay una segunda pestaña llamada **MainWindow.xaml.vb**, que corresponde al contenedor



EXTENSIBILIDAD DE XAML

Visual Studio 2010 se compone de varios lenguajes de programación, como Visual C#. En muchos casos, varias empresas suelen migrar sus proyectos desde Visual Basic hacia C#. Si el proyecto es del tipo WPF, solo habrá que rescribir el código fuente, dado que la UI podrá importarse sin ningún problema, por ser un lenguaje ajeno a VB o C#.

del código fuente que escribiremos para que nuestra aplicación funcione; este se guardará en el archivo con el mismo nombre. Volviendo a la primera pestaña, veremos que está dividida en dos. En la parte superior se muestra el área donde diseñaremos la **UI** de nuestro proyecto, y en el extremo inferior están los **tags XAML** que conforman la interfaz de la aplicación. En el lateral izquierdo veremos el **Cuadro de herramientas**, donde se listan todos los controles WPF que podemos utilizar en nuestra app; y sobre el lateral derecho, se ubican la ventana de **Propiedades** y el **Explorador de Soluciones**.

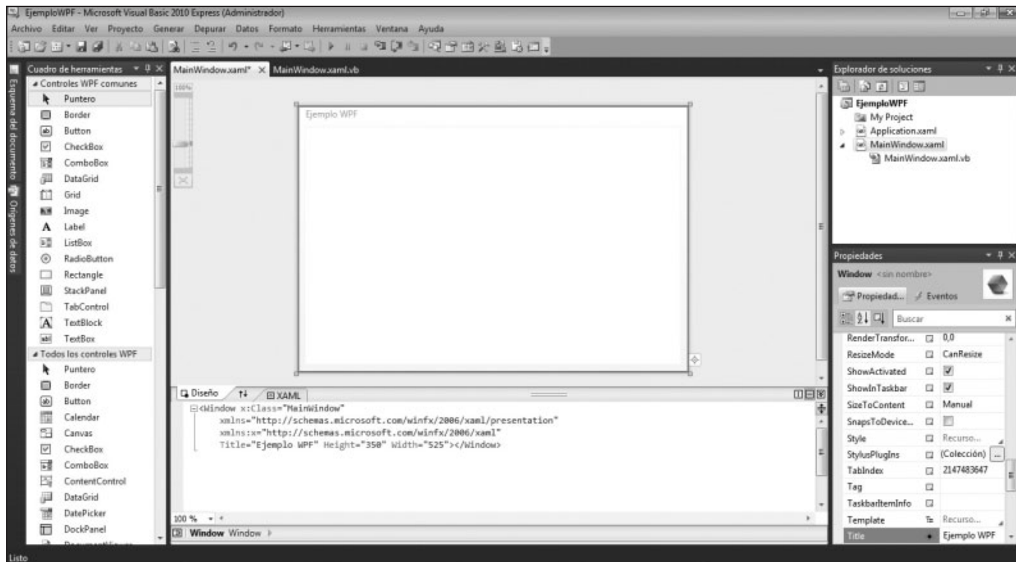


Figura 2. *El IDE de desarrollo Visual Basic 2010 de nuestros proyectos WPF tiene la misma estructura que el IDE de los proyectos Win Forms.*

La propiedad **Title** de **MainWindow** nos permite especificar el título del WPF Form; vamos a cambiarlo por **Ejemplo WPF**. Dentro del proyecto agregaremos algunos controles y configuraremos sus propiedades. Para ganar tiempo y aprender a agregar controles mediante código XAML, realizaremos este paso desde el panel XAML. Agregamos desde este panel el siguiente código:



IMÁGENES Y EFECTOS

WPF nos permite aplicar efectos especiales a los controles e imágenes utilizados en el proyecto. Si insertamos una imagen cualquiera, mediante el **tag <Border.BitmapEffect></Border.BitmapEffect>**, podremos aplicarle efectos 3D, con la posibilidad de aplicar las propiedades **DropShadowBitmap Effect Opacity**, **ShadowDepth**, **Softness**, **CenterY**, **AngleX** y **AngleY**.

```
...
Title="Ejemplo WPF" Height="350" Width="525">
<StackPanel>
    <Grid>
        <Label Content="Para:" Height="28" HorizontalAlignment="Left"
Margin="20,52,0,0" Name="Label1" VerticalAlignment="Top" />
        <Label Content="Asunto:" Height="28" HorizontalAlignment="Left"
Margin="20,78,0,0" Name="Label2" VerticalAlignment="Top" />
        <Label Content="Mensaje:" Height="28" HorizontalAlignment="Left"
Margin="20,108,0,0" Name="Label3" VerticalAlignment="Top" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="76,52,0,0"
 Name="TextBox1" VerticalAlignment="Top" Width="326" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="76,80,0,0"
 Name="TextBox2" VerticalAlignment="Top" Width="420" />
        <TextBox Height="195" HorizontalAlignment="Left"
Margin="76,108,0,0" Name="TextBox3" VerticalAlignment="Top" Width="421"
HorizontalScrollBarVisibility="Disabled"
VerticalScrollBarVisibility="Visible" CharacterCasing="Normal"
TextWrapping="Wrap" />
            <Button Content="Enviar" Height="64"
HorizontalAlignment="Left" Margin="420,11,0,0" Name="Button1"
VerticalAlignment="Top" Width="76" />
        </Grid>
    </StackPanel>
</Window>
        <TextBox Height="23" HorizontalAlignment="Left" Margin="76,52,0,0"
Name="TextBox1" VerticalAlignment="Top" Width="326" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="76,80,0,0"
Name="TextBox2" VerticalAlignment="Top" Width="420" />
        <TextBox Height="195" HorizontalAlignment="Left"
Margin="76,108,0,0" Name="TextBox3" VerticalAlignment="Top" Width="421"
```



MICROSOFT SILVERLIGHT

En las versiones **Professional** o superiores de Visual Studio 2010 se pueden crear tres tipos de proyectos relacionados con Silverlight: **Silverlight Application**, **Silverlight Class Library** y **Silverlight Navigation Application**. También vale destacar que Microsoft Silverlight tiene una integración total con los desarrollos de aplicaciones WPF.

```
HorizontalScrollBarVisibility=”Disabled” VerticalScrollBarVisibility=”Visible”
 CharacterCasing=”Normal” TextWrapping=”Wrap” />
            <Button Content=”Enviar” Height=”64” HorizontalAlignment=”Left”
Margin=”420,11,0,0” Name=”Button1” VerticalAlignment=”Top” Width=”76” />
```

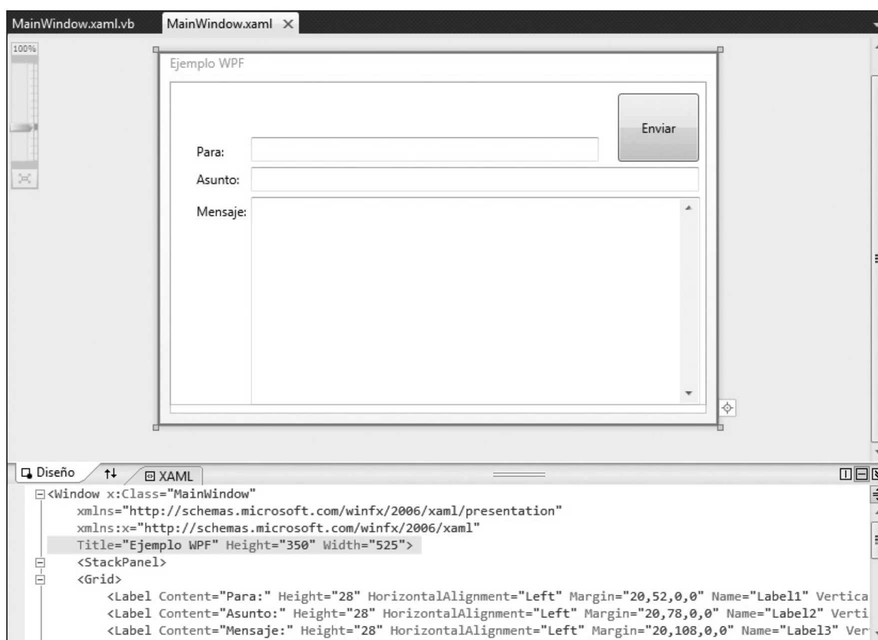


Figura 3. *Este es el resultado que veremos sobre la base del código XAML ingresado.*

Hasta ahora hemos desarrollado solamente la interfaz gráfica de la aplicación WPF. Su objetivo es enviar un correo electrónico hacia algún destino, utilizando una cuenta de **Gmail**. En caso de que dispongamos de una cuenta en **Hotmail**, deberemos proveer la configuración correspondiente. Para otros casos, habrá que averiguar el **host**, el **puerto** y la configuración necesaria. Ya finalizada la creación de la interfaz **UI**, procedemos a agregar el código que permitirá enviar nuestro mensaje. Completamos los campos **Para**, **Asunto** y **Mensaje**, y luego lo enviamos.

XAML MÁS ALLÁ DE VISUAL STUDIO

El lenguaje XAML es un lenguaje de marcado, con lo cual aprender a desarrollar UI a través de las etiquetas en vez de utilizar herramientas visuales puede abrirnos a un mercado muy amplio, ya que Adobe también tiene sus herramientas de desarrollo de aplicaciones de escritorio o web basadas en RIA.

Para agregar el código que hará funcionar la aplicación, hacemos clic con el botón derecho del mouse sobre el control **Button1**, y del menú contextual seleccionamos **Ver Código**. Se abrirá el archivo que almacena el código de nuestra aplicación XAML. Al principio de él, debemos importar los **namespaces** correspondientes para que la aplicación pueda trabajar con los protocolos de red. Agregamos al inicio del archivo **MainWindow.xaml.vb** los **namespaces System.Net** y **System.Net.Mail**, tal como vemos a continuación:

```
Imports System.Net
Imports System.Net.Mail

Class MainWindow
...
```

Ya dentro de la clase **MainWindow**, declaramos la variable **mensaje**, que será del tipo **MailMessage**, y **smtp**, que será del tipo **SmtpClient**:

```
Class MainWindow
    Dim mensaje As New MailMessage
    Dim smtp As New SmtpClient
```

Por último, solo queda escribir el código dentro del evento **Click()** de **Button1**. Dicho código nos permitirá enviar el mensaje redactado al destinatario especificado en **TextBox1**. Dentro del evento **Button1_Click(...)** escribimos el siguiente código:

```
...
With mensaje
 .From = New MailAddress("ferproonline@gmail.com")
 .To.Add(Trim(TextBox1.Text))
```



INFORMACIÓN DE LA WPF APPLICATION

Al igual que en las aplicaciones de escritorio **Windows Form**, cualquier tipo de desarrollo que sea llevado a cabo con **WPF** puede contener múltiples WPF **Window**. También podemos acceder a la información necesaria sobre la aplicación en tiempo de ejecución, que puede ser consultada utilizando la clase **My.Application**.

```
  .Subject = Trim(TextBox2.Text)
  .Priority = MailPriority.Normal
  .Body = Trim(TextBox3.Text)
End With
...
```

En principio, utilizamos **With mensaje** para establecer las distintas propiedades de mensaje sin repetir varias veces el nombre del objeto. La propiedad **From** permite especificar la dirección de correo electrónico de quien está originando el mensaje. La propiedad **To** va acompañada de la función **Add()**, que permite agregar una o más direcciones de correo (en caso de que deseemos enviar un único mensaje a varias direcciones, debemos utilizar una variable de tipo **String**, donde almacenamos **la cadena de colección**; dichos mensajes deberán estar separados por coma o punto y coma, y deberemos tratarlos con la función **Split()**). **Subject** permite especificar el **Asunto** del mensaje, **Priority** establece si este es urgente o no, y **Body** contendrá el texto del cuerpo del mensaje. Luego incluimos la segunda parte del código, justo debajo de lo escrito hasta el momento:

```
Try
     With smtp
          .EnableSsl = True
          'Puertos de Hotmail y Gmail.
          .Port = "587"
          'Servidor Hotmail = smtp.live.com
          .Host = "smtp.gmail.com"
          .Credentials = New NetworkCredential("TuUsuarioDeCorreo", "Aquí
ingresa tu clave")
          .Send(mensaje)
          MsgBox("Mensaje enviado con éxito.", MsgBoxStyle.OkOnly, "Envío de
Email WPF")
```



CÓDIGO FUENTE WPF

El código fuente utilizado en estos ejemplos es exactamente igual al empleado en el desarrollo web y en el desarrollo basado en Windows Forms. Por lo tanto, si hemos logrado una muy buena práctica en los otros, desarrollar aplicaciones WPF será algo más que sencillo para nosotros.

```
    End With
Catch ex As Exception
    MsgBox("Error en el envío del mensaje." & Chr(13) & ex.Message,
MsgBoxStyle.OkOnly, "Error en envío de email")
End Try
...
```

En esta segunda parte utilizamos **Try Catch** para manipular y anunciar si hubo un error en el intento de envío del mensaje de correo electrónico. Repetimos también el uso de la sentencia **With** junto con **smtp**.

En este ejemplo, utilizamos un **servidor SMTP** propio de **Gmail**. Si tenemos cuenta de **Hotmail**, dentro de este código encontraremos comentado el **Host** necesario para que la prueba funcione. Ambos servidores de correo electrónico aquí mencionados utilizan el puerto **587** para enviar mensajes de correo electrónico salientes, y la conexión se realiza mediante **SSL seguro**.

```
Imports System.Net
Imports System.Net.Mail

Class MainWindow
    Dim mensaje As New MailMessage
    Dim smtp As New SmtpClient

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs) Handles B
        With mensaje
            .From = New MailAddress("ferproonline@gmail.com")
            .To.Add(Trim(TextBox1.Text))
            .Subject = Trim(TextBox2.Text)
            .Priority = MailPriority.Normal
            .Body = Trim(TextBox3.Text)
        End With
        Try
            With smtp
                .EnableSsl = True
                .Port = "587" 'Puertos de Hotmail y Gmail.
                .Host = "smtp.gmail.com" 'Servidor Hotmail = smtp.live.com
                .Credentials = New NetworkCredential("TuUsuarioDeCorreo", "Aquí ingresa tu clave")
                .Send(mensaje)
                MsgBox("Mensaje enviado con éxito.", MsgBoxStyle.OkOnly, "Envío de Email WPF")
            End With
        Catch ex As Exception
            MsgBox("Error en el envío del mensaje." & Chr(13) & ex.Message, MsgBoxStyle.OkOnly, "Error en envío d
        End Try
    End Sub
```

Figura 4. *El código que da vida a nuestra aplicación WPF es simple y similar al del resto de las aplicaciones de escritorio que podemos crear con Visual Basic 2010.*



VENTAJAS DE LAS APLICACIONES RIA

Las aplicaciones RIA combinan las ventajas de una aplicación web con las de una aplicación tradicional de escritorio. Si bien las aplicaciones web poseen poca capacidad multimedia, las aplicaciones **XAML Apps** aún no permiten integrar reproducción de video de manera nativa, con lo cual hay que contar con un reproductor externo.

Por el momento, esto es todo el desarrollo de la aplicación. Solo nos resta probar su correcto funcionamiento, para lo cual ejecutamos el proyecto presionando **F5** y llenamos los campos para verificar el que todo marche bien.

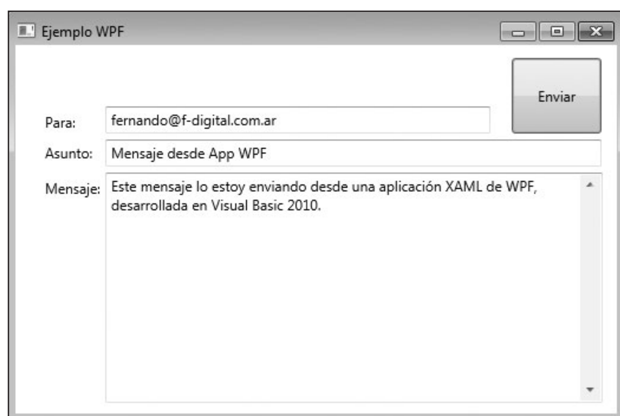


***Figura 5.** Debemos completar los campos de la app WPF antes de poder enviar el correo electrónico a su destinatario.*

Enviamos el mensaje redactado presionando el botón **Enviar**. Si hemos ingresado nuestro **usuario** y **password** correctamente, el mensaje será despachado y se nos avisará mediante un **MessageBox** que todo salió correctamente. **Try Catch** permitirá que trabajemos con cuidado ante cualquier error que se produzca. Para probarlo, debemos cambiar el **Nombre de usuario** o la **contraseña** dentro del evento **Click()** de **Button1**. Esto provocará un error forzado, que será controlado por **Catch** y visualizado en pantalla a través del **MessageBox**. Volvamos a ejecutar nuestro proyecto para probarlo. En la **Figura 6** podemos observar que se provoca un error.

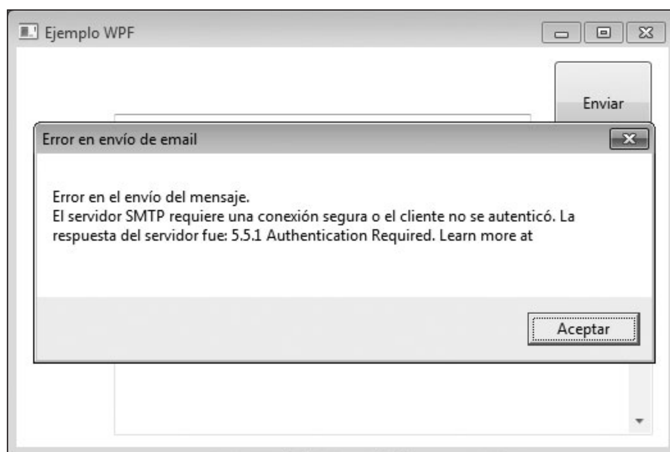


***Figura 6.** Este error fue forzado, debido al cambio de **username** o **password**, que no pudo validarse en el **servidor SMTP**.*

HERRAMIENTAS MICROSOFT DE DESARROLLO XAML

El desarrollo de aplicaciones XAML no se limita a correrlas sobre el escritorio: también es posible desarrollar **aplicaciones XAML Web** desde VB 2010. La metodología de desarrollo es similar a la que hemos visto en nuestro ejemplo anterior, con la salvedad de que, al momento de ejecutarla, esta se iniciará en una solapa de **Internet Explorer**. El requisito para ejecutar aplicaciones Web XAML es disponer de la versión 8 o superior del navegador de **Microsoft**.

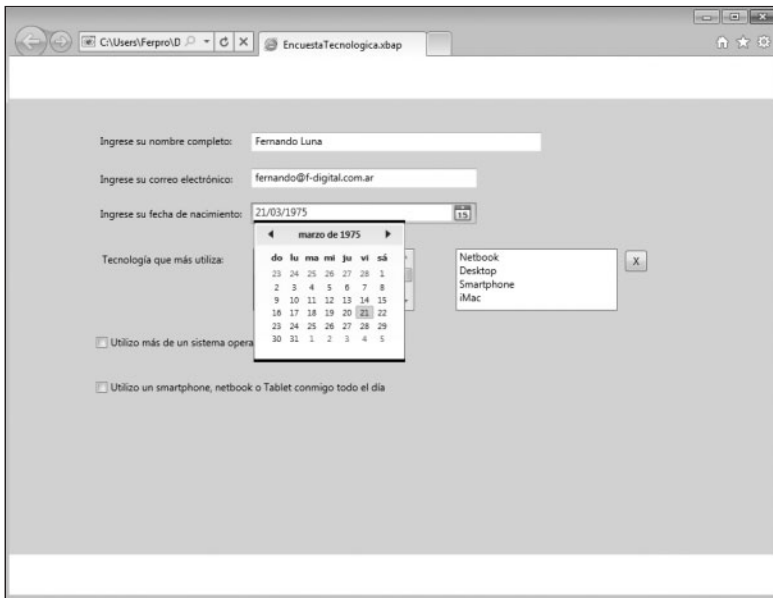


***Figura 7.** Nuestra aplicación Web WPF contiene una interfaz agradable que combina la estética de una aplicación Windows Forms con una aplicación Web.*

Para complementar el desarrollo de aplicaciones XAML, y con una fuerte orientación al aspecto estético, Microsoft lanzó hace unos años la suite **Microsoft**



APLICACIONES WPF

La última versión de **Microsoft Windows Messenger** para **Windows Vista** o **Seven**, al igual que otras aplicaciones que componen la **Suite Microsoft Windows Live**, han sido desarrolladas íntegramente utilizando las ventajas que presenta WPF. Esto explica que el desarrollo de soluciones bajo este tipo de aplicaciones no tenga límites.

Expression Blend, actualmente en su versión 4.0. Esta suite combina desarrollos de aplicaciones web y de escritorio, aplicaciones para **Windows Phone 7** (que se verán en el **Apéndice B** de este libro), y aplicaciones o juegos del tipo **XNA**, para consolas **Microsoft Xbox 360**.

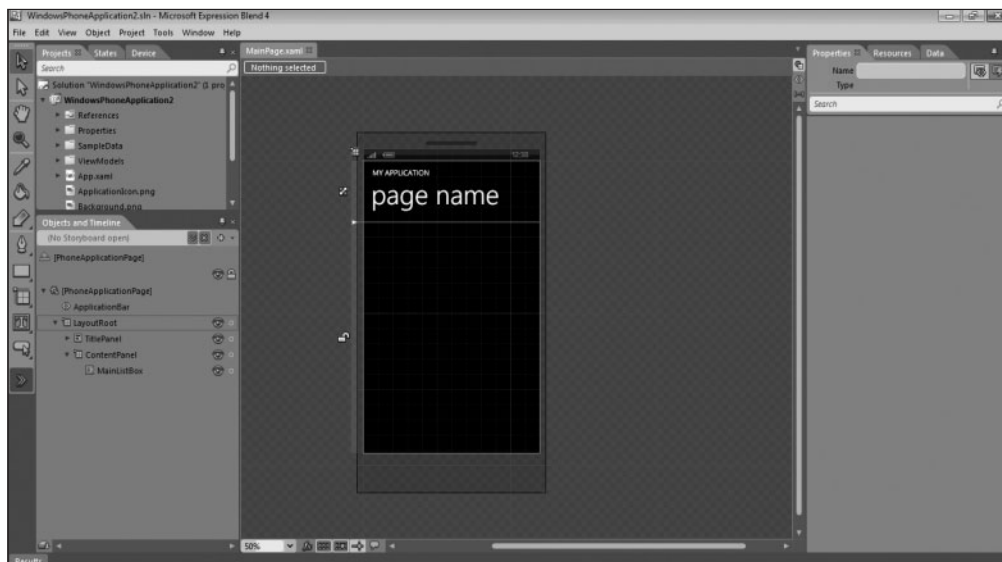


Figura 8. *Microsoft Expression Blend permite combinar el desarrollo de aplicaciones web con aplicaciones XAML, tanto para Windows de PC como para Windows Phone 7.*

HERRAMIENTAS DE TERCEROS PARA DESARROLLAR XAML

El desarrollo de aplicaciones XAML por parte de Microsoft nace a partir de que **Adobe Air** y **Adobe Flex** llegaron al mercado y no pararon de crecer, cosechando un exito importante en este campo. A su vez, estos fueron ideados por la constante exigencia de usuarios que querían utilizar interfaces de aplicaciones diseñadas en **Flash**, pero que no necesitaban tanta complejidad como la requerida en Flash para un diseño UI práctico. Cualquier aplicación del tipo XAML desarrollada en otro entorno también podrá integrarse sin problema alguno en nuestro proyecto Windows Presentation Foundation de Visual Studio. **Adobe Flash Builder** también permite crear aplicaciones RIA con el estilo Flash. Está disponible para adquirir de manera individual, y también forma parte de la suite **Premium Adobe CS5**.

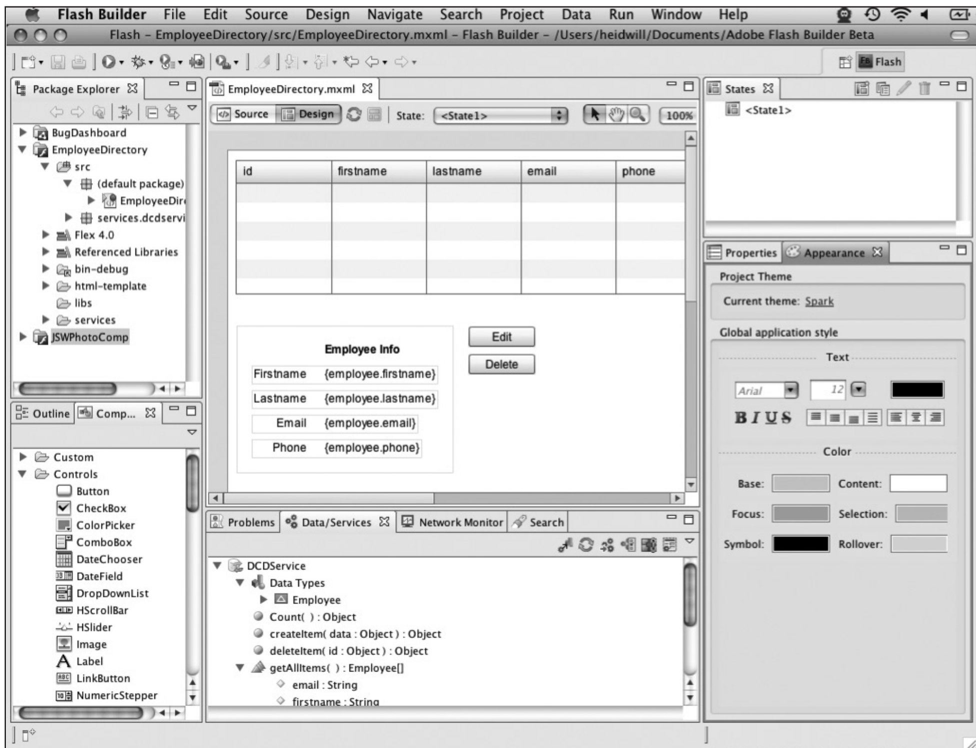


***Figura 9.** Adobe Flash Builder pasó de ser un simple programa de animación a ser un completo GUI que permite desarrollar aplicaciones basadas en XML.*

... RESUMEN

Hasta aquí hemos hecho un repaso generalizado de las alternativas propuestas por Microsoft para desarrollar aplicaciones de escritorio o web basadas en RIA. Rich Internet Applications es tomado como el futuro del desarrollo de software bajo el entorno Windows, con lo cual resulta importante profundizar más en las distintas características que brinda. Windows Phone 7 cambió completamente el modelo de desarrollo de sus aplicaciones móviles al migrar hacia el formato RIA, y es probable que en el transcurso de las próximas versiones del sistema de escritorio Windows veamos un cambio similar.



ACTIVIDADES

PREGUNTAS TEÓRICAS

1. ¿Qué es una aplicación XAML?

2. ¿Cuántos tipos de desarrollos WPF pueden llevarse a cabo con Visual Studio 2010 Express Edition?

3. ¿Las aplicaciones WPF guardan su diseño de Front End y el código para su funcionamiento en un único archivo?

4. ¿Se puede ejecutar indistintamente una misma aplicación WPF tanto en Windows como en un navegador web?

5. ¿Qué suite de aplicaciones conocida fue desarrollada íntegramente en WPF?

EJERCICIOS PRÁCTICOS

1. Investigue si es posible adjuntar archivos al envío de un correo electrónico mediante la aplicación WPF creada en este capítulo.

2. Agregue los campos CC y CCO, y modifique el código de la aplicación para que soporte este pequeño cambio.

3. Agregue, a continuación, una imagen al proyecto y utilice los tags correspondientes para aplicarle efectos.

4. Cree un algoritmo básico dentro de una función que le permita revisar el contenido de los campos de dirección de correo electrónico, antes de que este sea enviado.

5. Investigue acerca de cómo utilizar efectos sobre las imágenes dentro de WPF.

Aplicaciones prácticas en pocos clics

Este apéndice está dedicado, íntegramente, a realizar aplicaciones prácticas con muy pocos componentes y escasas líneas de código. Estudiaremos, entre ellos, el componente WebBrowser, con sus propiedades, métodos y eventos; y algunos controles básicos que ya hemos utilizado en los ejemplos a lo largo de este libro. También repasaremos algunos de los servicios que pone a nuestra disposición la librería wmp.dll.

COMPONENTES ÚTILES QUE FACILITAN LOS DESARROLLOS

La mayoría de las herramientas de desarrollo que Microsoft nos brinda pueden acceder a **librerías**, **bibliotecas de clase** y componentes **DLL** alojados en nuestro sistema operativo, para aprovecharlos en el desarrollo de nuestras aplicaciones. Este servicio es una gran ventaja para cualquier programador, dado que permite ahorrar miles de líneas de código y muchas horas de investigación.

Esto es lo que, en el mundo de la programación, se conoce como reutilización de componentes. Sin perder más tiempo, pongamos manos a la obra e iniciemos el desarrollo de aplicaciones que nos permitan aprovechar las herramientas existentes en el sistema operativo de Microsoft,

El control WebBrowser

Este componente se encuentra dentro del espacio de nombres **System.Windows.Forms**. Es una clase que pertenece al control del navegador de **Internet Explorer**, con lo cual podemos utilizarlo de manera independiente al navegador, pero aprovechando todos los **plugins** y configuraciones básicas que se hayan hecho dentro del browser de Microsoft.

Aplicaciones externas como el plugin **Adobe Flash Player**, **JavaScript** y el plugin de **Windows Media Player** funcionarán sin que tengamos que realizar ninguna intervención dentro de nuestro próximo proyecto. Este control es empleado por algunos navegadores alternativos que encontramos en la Web, como **SlimBrowser**, **Avant Browser** o **WebbIE**, además de otros ya discontinuados, como **NeoPlanet** y **UltraBrowser**.

Incluso, hasta podemos combinar un desarrollo de **aplicaciones Web ASP.NET** con una **aplicación Windows Form** que directamente abra la correspondiente web en una ventana independiente, sin que el usuario deba ingresar direcciones web o impidiendo que vea el URL desde donde se está cargando el sitio. De esta manera, estamos aplicando una medida más de seguridad a nuestros desarrollos.



REUTILIZACIÓN DE ANTIGUOS CONTROLES

Si disponíamos de controles **ActiveX** que potenciaban nuestras aplicaciones desarrolladas en anteriores versiones del lenguaje, podemos seguir utilizándolos en esta versión de Visual Basic. Solo debemos agregarlos desde el **Cuadro de herramientas**, haciendo clic con el botón derecho y seleccionando **Elegir elementos/Pestaña Componentes COM**.

Ahora sí, tomemos la iniciativa y pongámonos manos a la obra para crear con unas pocas líneas un navegador web propio.

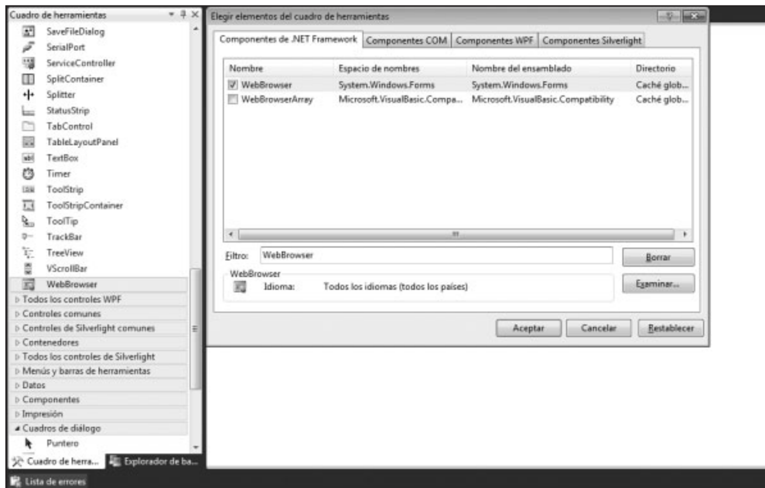


***Figura 1.** Si no encontramos el componente cargado entre los controles, podemos agregarlo a través del menú contextual, en la opción **Elegir Elementos...** del **Cuadro de herramientas**.*

Desarrollo de WNetBrose, un navegador web

Desarrollaremos a continuación un navegador web utilizando el componente **WebBrowser**. El navegador contendrá una interfaz **SDI**. Podemos modificarlo luego para que pueda contener múltiples pestañas, como la mayoría de los navegadores actuales, ya sea a través del uso del control **TabControl**, o usando un **MDIForm** y creando nuevas instancias del **Form** del navegador.

Estructura de la interfaz

Iniciamos un nuevo proyecto **Aplicación Windows Form**. Utilizaremos en él un **Form**, cuya propiedad **Name** será **frmBrowser**, cuatro controles **Button**, un control **ProgressBar**, un **ComboBox**, un **ToolTip**, un **Label**, tres **Panel** y un **SplitContainer**. Agregamos como primer control un **SplitContainer** y ajustamos sus propiedades según las especificaciones de la **Tabla 1**.

CONTROL SPLITCONTAINER	
PROPIEDAD	VALOR
Name	SplitContainer1
Orientation	Horizontal
Dock	Fill
FixedPanel	Panel1

***Tabla 1.** Ajuste de propiedades del control **SplitContainer1**.*

Luego agregamos dentro del contenedor **SplitContainer1.Panel1** los tres controles **Panel** y configuramos sus propiedades según las especificaciones de la **Tabla 2**.

CONTROL PANEL1	
PROPIEDAD	VALOR
Name	Panel1
Dock	Right
Control Panel2	
Propiedad	Valor
Name	Panel2
Dock	Left
Control Panel3	
Propiedad	Valor
Name	Panel3
Dock	Fill
AutoSizeMode	GrowAndShrink

***Tabla 2.** Ajuste de propiedades de los controles Panel, que contendrán los controles básicos del navegador.*

Dentro del contenedor **SplitContainer1.Panel2** agregamos el control **WebBrowser** y configuramos sus propiedades según la **Tabla 3**.

CONTROL WEBBROWSER	
PROPIEDAD	VALOR
Name	WebBrowser1
Anchor	Top, Bottom, Left, Right
AllowNavigation	True
ScriptErrorsSuppressed	True
ScrollBarsEnabled	True

***Tabla 3.** Ajuste de propiedades del control WebBrowse.*

Dentro de **Panel1** agregamos los controles **Button** y el control **ProgressBar**, y configuramos sus propiedades básicas según se indica en la **Tabla 4**.

CONTROL BUTTON1	
PROPIEDAD	VALOR
Name	btnGoBack
Font	Webdings; 14.25pt
Size	32; 32
Text	3
ToolTip	Volver a la página anterior

CONTROL BUTTON1	
PROPIEDAD	**VALOR**
Control Button2	
Name	btnGoForward
Font	Webdings; 14.25pt
Size	32; 32
Text	4
ToolTip	Ir a la página siguiente
Control Button3	
Name	btnReload
Font	Webdings; 14.25pt
Size	32; 32
Text	q
ToolTip	Recargar página
Control Button4	
Name	btnStop
Font	Webdings; 14.25pt
Size	32; 32
Text	r
ToolTip	Detener carga de página
Control ProgressBar	
Name	ProgressBar1
Maximun	0
Minimun	0
Value	0
Size	32; 32
Style	Marquee

***Tabla 4.** Ajuste de propiedades de los controles Button y ProgressBar.*

Dentro de **Panel3** agregamos un control **ComboBox** ajustando sus propiedades **Name = comboURL** y **Anchor = Left, Right**. Por último, agregamos un **Label** dentro de **Panel2** y ajustamos sus propiedades **Anchor = Top, Bottom, Left, Right**, y **Text = URL:**.



MANIPULAR VIDEOS DESDE VB

Así como desarrollamos una aplicación utilizando la librería wmp.dll, podemos aprovecharla para incluir la posibilidad de reproducir videos. En los métodos, propiedades y funciones de dicho objeto, encontraremos referencias a cada una de las funciones necesarias para la reproducción. Este componente usa los códecs registrados en nuestro sistema.

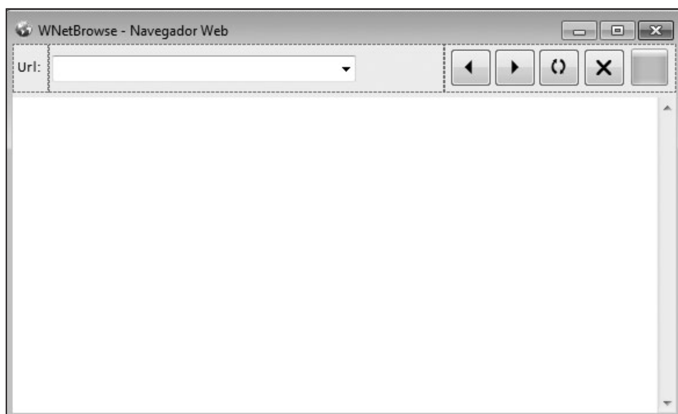


Figura 2. *Para dar un orden similar a la pantalla de los web browsers conocidos, realizamos la distribución de los controles de nuestro proyecto tomando como ejemplo la* ***Figura 2****.*

Codificación del browser

Una vez distribuida la disposición de los controles dentro de **frmBrowser**, procedemos a agregar el código para que nuestro navegador web funcione. En principio, creamos dos procedimientos públicos: uno ejecutará la acción de navegar hacia una **URL** específica, y el otro agregará las **URLs** navegadas a nuestra **colección item** del control **comboURL**, siempre y cuando esta URL no haya sido agregada o navegada previamente. De este modo, dispondremos de un historial de navegación mientras nuestro **web browser** esté abierto.

```
Public Class frmBrowser
    Private Sub NavegoaURL()
        If Trim(ComboURL.Text) <> "" Then
            WebBrowser1.Navigate(Trim(ComboURL.Text))
        End If
    End Sub

    Private Sub agregoURL(ByVal url As String)
        Dim bExiste As Boolean
        bExiste = False
        If ComboURL.Items.Count - 1 > -1 Then
            For i = 0 To (ComboURL.Items.Count) - 1
                If ComboURL.Items(i) = Trim(url) Then
                    bExiste = True
                    Exit Sub
                End If
            Next
```

```
        End If
        ComboURL.Items.Add(Trim(ComboURL.Text))
    End Sub
…
```

Luego agregamos el resto del código que controlará las funciones de nuestro browser. Para hacerlo, identificamos dentro del siguiente código el nombre de cada uno de los controles añadidos al programa:

```
…
Private Sub ComboURL_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles ComboURL.KeyDown
        If e.KeyCode = Keys.Enter Then
            NavegoaURL()
        End If
End Sub

Private Sub ComboURL_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ComboURL.SelectedIndexChanged
        NavegoaURL()
End Sub

Private Sub WebBrowser1_DocumentCompleted(ByVal sender As System.Object,
ByVal e As System.Windows.Forms.WebBrowserDocumentCompletedEventArgs)
Handles WebBrowser1.DocumentCompleted
        ProgressBar1.Maximum = 0
End Sub

Private Sub WebBrowser1_Navigated(ByVal sender As Object, ByVal e As
System.Windows.Forms.WebBrowserNavigatedEventArgs) Handles WebBrowser1.Navi
gated
        ComboURL.Text = WebBrowser1.Url.ToString
        If WebBrowser1.CanGoBack Then btnGoBack.Enabled = True Else btnGo
Back.Enabled = False
        If WebBrowser1.CanGoForward Then btnGoForward.Enabled = True Else
btnGoForward.Enabled = False
        agregoURL(ComboURL.Text)
        btnStop.Enabled = False
        btnReload.Enabled = True
```

```
End Sub

Private Sub WebBrowser1_Navigating(ByVal sender As Object, ByVal e As
System.Windows.Forms.WebBrowserNavigatingEventArgs) Handles
WebBrowser1.Navigating
        ProgressBar1.Maximum = 100
        Me.Text = "WNetBrowse - Navegador Web - " & WebBrowser1.Document
Title
        If WebBrowser1.CanGoBack Then btnGoBack.Enabled = True Else btnGo
Back.Enabled = False
        If WebBrowser1.CanGoForward Then btnGoForward.Enabled = True Else
btnGoForward.Enabled = False
        btnStop.Enabled = True
        btnReload.Enabled = False
End Sub

Private Sub btnGoBack_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGoBack.Click
        WebBrowser1.GoBack()
End Sub

Private Sub btnGo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGoForward.Click
        WebBrowser1.GoForward()
End Sub

Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnReload.Click
        WebBrowser1.Refresh()
End Sub

Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnStop.Click
        WebBrowser1.Stop()
End Sub
End Class
```

Del control **WebBrowser1** utilizamos los eventos **Document_Completed()**, que nos permite saber cuándo ha finalizado la carga de una página; **Navigated()**, que nos indica si se ha podido navegar hacia ella; y **Navigating()**, que nos permite ejecutar

determinadas acciones mientras la página se está cargando, como animar el control **ProgressBar** para que indique que el navegador web está trabajando.
Las funciones **Goback()** y **GoForward()** permiten que nos desplacemos a la página anterior o a la siguiente que hayan sido navegadas. **CanGoBack()** y **CanGoForward()** se utilizan para conocer si se puede navegar hacia adelante o hacia atrás, para habilitar o no el estado de los botones **BtnGoBack** y **BtnGoForward**.
Refresh() y **Stop()** son dos de las principales funciones destinadas a recargar la página donde nos encontramos o detener la carga de la página que se está cargando en ese momento. La propiedad **Navigate()** ejecuta la acción de navegar hacia la URL que indicamos en **comboURL**. La propiedad **DocumentTitle** especifica el título de la página web donde nos encontramos.



Figura 3. WNetBrowser ya tiene el código necesario para funcionar. Veámoslo en acción.

Ejecutamos nuestro proyecto con **F5** e ingresamos una URL, luego de lo cual presionamos **Enter**. Veremos cómo se altera el estado **Enabled** de los botones y la animación de **ProgressBar1** mientras se carga la página y cuando termina.



USO DEL CONTROL WEB

En nuestra aplicación WNet Browser utilizamos una clase que toma como base el motor del navegador Internet Explorer. Así como usamos el engine del navegador de Microsoft, también disponemos de todos los componentes y plugins instalados en este navegador, con lo cual no debemos hacernos problema, por ejemplo, por el plugin de Flash Player.

Luego, ingresamos otra dirección distinta de la actual, para que se genere un historial de navegación y se habiliten los botones de desplazamiento.



Figura 4. *El historial URL de nuestro web browser comienza a acumular las páginas visitadas.*

LA LIBRERÍA WMP.DLL

Windows contiene la librería **wmp.dll**, encargada de manejar todo lo referido a medios de audio y video, de manera ya sea local o remota. Es utilizada por el reproductor de Microsoft, Windows Media Player, y al encontrarse en casi todos los equipos que corren Windows en la actualidad, también podemos usarla aprovechando sus propiedades, métodos y eventos.

Desarrollo de ROL Player

A continuación, desarrollaremos una aplicación que nos permitirá sintonizar radios a través de **streaming**. El objetivo de este programa será almacenar en un **archivo de texto** nuestras radios preferidas y disponer de ellas para poder escuchar música cuando queramos, sin tener que ingresar en una web determinada y ejecutar el reproductor online.

Internet cada vez se enriquece más con streaming de música, con lo cual podremos sacar provecho y guardar los que más nos interesen para escucharlos en todo momento con nuestro reproductor.

Estructura de la interfaz

Creamos un nuevo proyecto **Aplicación Windows Forms**. En **Form1** agregamos cuatro controles **Label**, dos **Button**, un **TrackBar** y un **ListBox**. Para darle el aspecto de **Panel LCD**, al ejemplo que veremos en las capturas de pantalla se incorporó una imagen de fondo en **Form1**.

Podemos crear algo similar de las mismas dimensiones que el **Form** con cualquier editor de imágenes que tengamos instalado, o directamente, obviar este fondo. A continuación, ajustamos las propiedades de los controles que utilizaremos. En primer lugar, vamos a configurar los labels según se muestra en la **Tabla 5**. Para este ejemplo utilicé Adobe Fireworks, también puede sernos útil **GIMP** o **Paint.Net**, dos editores de imágenes gratuitos.

CONTROL LABEL1	
PROPIEDAD	VALOR
Name	Label1
BackColor	Transparent
Font	LCD2; 20.25pt
ForeColor	Black
Text	Seleccione emisora
Control Label2	
Propiedad	Valor
Name	Label2
BackColor	Transparent
Font	LCD2; 12pt
ForeColor	Black
Text	Estado de conexión
Control Label3	
Propiedad	Valor
Name	Label3
BackColor	Transparent
Font	Microsoft Sans Serif; 6.75pt
ForeColor	Black
Text	Volumen
Control Label4	
Propiedad	Valor
Name	Label4
BackColor	Transparent
Font	LCD2; 12pt
ForeColor	Black
Text	Universal Resource Locator

Tabla 5. *Ajuste de propiedades de los controles Label.*

La fuente **LCD utilizada** es gratuita, y puede descargarse desde **www.fonts4free.net/lcd-lcd-mono-font.html**. Ahora vamos a configurar los controles **ListBox**, **TrackBar**, **Timer1** y los **Button** con las especificaciones que se indican en la **Tabla 6**.

CONTROL LISTBOX	
PROPIEDAD	VALOR
Name	lbEmisoras
BackColor	ControlDark
ForeColor	White
ToolTip	Seleccione la emisora
Control Button1	
Propiedad	Valor
Name	btnEmisoras
Font	Webdings; 20.25pt
Text	»
ToolTip	Agregar emisora
Control Button2	
Propiedad	Valor
Name	btnEscuchar
Font	Webdings; 20.25pt
Text	-
ToolTip	Escuchar emisora seleccionada
Control TrackBar	
Propiedad	Valor
Name	trackVol
Maximun	100
Minimun	0
SmallChange	1
ToolTip	Volumen
Control Timer	
Propiedad	Valor
Interval	10

CORRECTO USO DE WMPOPENSTATE

WMPOpenState permite definir, mediante tipos enumerados, los posibles distintos estados del reproductor Windows Media Player, y por consiguiente, de la librería **wmp.dll**. Armando un script minucioso y correcto, podemos tener la información en detalle sobre el estado por el cual cursa dicha librería en cada momento de su uso.

CONTROL LISTBOX	
PROPIEDAD	VALOR
Enabled	False
Name	Timer1

***Tabla 6.** Ajuste de las propiedades de los controles ListBox, Button, Timer y TrackBar.*



***Figura 5.** Nuestro proyecto WNet Browser muestra una UI organizada e intuitiva, la distribución de controles es importante para la identidad y el fácil manejo de nuestra app por parte de los usuarios finales.*

Es hora de poner a funcionar nuestro proyecto. Lo primero que haremos será crear un archivo de texto dentro de nuestro proyecto con el nombre **radiosonline.txt**. Para hacerlo, desde la barra de herramientas seleccionamos el botón **Agregar nuevo elemento...** y de la lista elegimos **Archivo de texto**. Dentro de este archivo agregamos algunas radios que conozcamos, con sus respectivas URL.

radiosonline.txt × Form1.vb Form1.vb [Diseño]

```
Urban Sound!http://streaming.metro951.com/metro
Rock And Roll!http://streaming.fmrockandpop.com/rockandpop
Rock hits!http://200.42.92.48:8111/Rock Hits 64.mp3?1273519709203.mp3
Party Dance!http://200.42.92.48/Party Dance 64.mp3?1274272654359.mp3
New Party Dance!http://200.42.92.48/New Party Radio 64.mp3?1274272693562.mp3
Radio Stone!http://200.42.92.48/Rollinga 64.mp3?1274272719640.mp3
```

***Figura 6.** El archivo radiosonline.txt contiene algunas emisoras que utilizaremos en nuestro proyecto radial.*

La carga de la radio va antecedida por el nombre descriptivo de la emisora, el símbolo **!** y la dirección correspondiente al streaming.

Codificación de las funciones

Anteriormente, hemos tratado detenidamente lo importante que es referencias los namespaces al inicio de cada clase. Por lo tanto, para comenzar, debemos agregar al proyecto las referencias **System.IO** y **WMPLib**, para poder utilizar la librería **DLL**:

```
Imports System.IO
Imports WMPLib
```

Luego declaramos las variables con la ruta del archivo de texto, una variable que indica si se seleccionó una emisora y la instancia a utilizar de **WMPLib**:

```
Public Class frmROLPlayer
    Public bSelected As Boolean = False
    Public wmpRadio As New WMPLib.WindowsMediaPlayer
    Public pathArchivo As String = "C:\Users\...ruta al archivo de texto...
 \ROLPlayer\radiosonline.txt"
…
```

Creamos dos procedimientos: el primero cargará las radios del archivo **radiosonline.txt**, y el segundo buscará el streaming correspondiente a la radio seleccionada:

```
Sub cargoRadios()
    Dim objStreamReader As StreamReader
    Dim strLine As String
    objStreamReader = New StreamReader(pathArchivo)
    strLine = objStreamReader.ReadLine
    lbEmisoras.Items.Clear()
    Do While Not strLine Is Nothing
       Dim Separa() As String
       Separa = Split(strLine, "!", 0, CompareMethod.Text)
       lbEmisoras.Items.Add(Separa(0))
       strLine = objStreamReader.ReadLine
    Loop
    objStreamReader.Close()
End Sub
Sub Selecciono Radio()
    Dim objStreamReader As StreamReader
    Dim strLine As String
    objStreamReader = New StreamReader(pathArchivo)
    strLine = objStreamReader.ReadLine
    Do While Not strLine Is Nothing
        Dim Separa() As String
        Separa = Split(strLine, "!", 0, CompareMethod.Text)
```

```
        If Separa(0) = lbEmisoras.Text Then
            Label1.Text = Separa(0)
            Label4.Text = Separa(1)
            objStreamReader.Close()
            bSelected = True
            Exit Sub
        End If
        strLine = objStreamReader.ReadLine
    Loop
    bSelected = False
End Sub
…
```

Por último, cargamos las funciones en el evento **Form_Load**, **Timer1_Tick**, **TrackVol_Scroll** y el control **btnEscuchar_Click**:

```
…
Private Sub frmROLPlayer_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    cargoRadios()
    trackVol.Value = wmpRadio.settings.volume
End Sub

Private Sub btnEscuchar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnEscuchar.Click
    SeleccionoRadio()
    If bSelected Then wmpRadio.URL = Label4.Text.ToString
    Timer1.Enabled = True
    wmpRadio.controls.play()
    trackVol_Scroll(sender, e)
```



ABRIR UNA PÁGINA CON IE

Si desde nuestro navegador queremos acceder a una determinada página, pero, a su vez, queremos abrirla con el navegador de Microsoft, **Internet Explorer**, y no con nuestro desarrollo, podemos usar la clase **Process.Start**, tal como lo indica el siguiente ejemplo: **Process.Start("iexplore.exe", www.redusers.com)**.

```
End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
     Dim s As String = wmpRadio.openState.ToString
     Select Case s
     Case WMPOpenState.wmposMediaLocating.ToString
     Label2.Text = “Localizando URL...”
     Case WMPOpenState.wmposMediaOpening.ToString
     Label2.Text = “Abriendo URL...”
     Case WMPOpenState.wmposMediaConnecting.ToString
     Label2.Text = “Conectando con emisora...”
     Case WMPOpenState.wmposMediaOpen.ToString
     Label2.Text = “Reproduciendo emisora”
     End Select
End Sub

Private Sub trackVol_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles trackVol.Scroll
     wmpRadio.settings.volume = trackVol.Value
     Label3.Text = “Volumen “ & trackVol.Value.ToString & “%”
End Sub
End Class
```

En **wmpRadio.URL** especificamos la dirección URL que deseamos reproducir. Dentro del control **Timer1** controlamos el estado de la propiedad **wmpRadio.WMPOpenState**, para saber si está buscando la URL, conectando o reproduciendo. El método **wmpRadio.Controls.Play()** reproduce lo ingresado en la propiedad URL.
trackVol.Value especifica, entre **0** y **100**, el valor que tendrá el volumen de la reproducción; dicho valor debe asignarse a la propiedad **wmpRadio.settings.volume**. Y a través de **StreamReader** leemos el contenido de **radiosonline.txt**.



TOOLBAR DE WNET BROWSER

La barra de herramientas de nuestro browser web es muy rudimentaria, por lo que podemos mejorarla para aprovechar las funciones vistas en el **Capítulo 4**, donde desarrollamos una barra de herramientas completa y funcional para nuestro editor de textos. Aprovechando estas funciones, mejoraremos la barra de nuestro browser web.

Ejecutamos nuestro proyecto presionando **F5** y seleccionamos una de las radios que cargamos al archivo de texto.

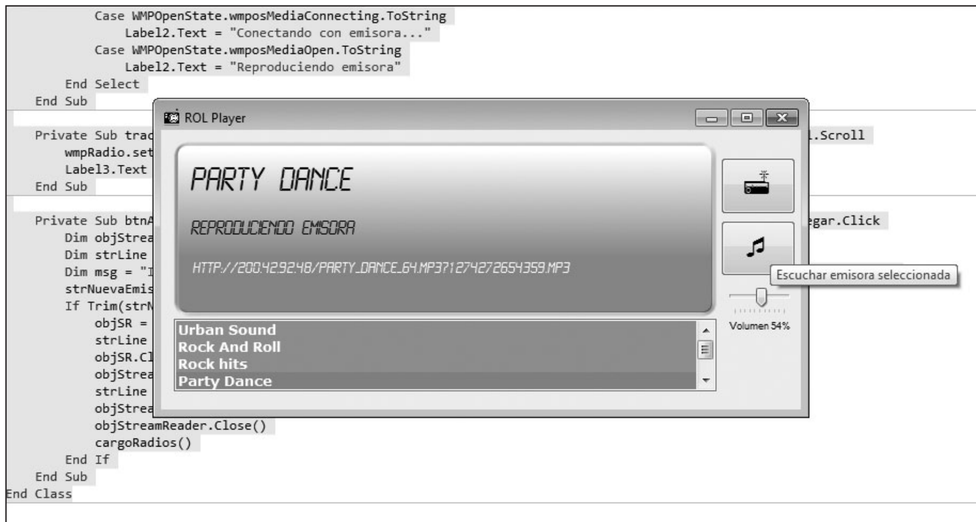


***Figura 7.** Nuestro reproductor de radios online en ejecución demuestra su sencilla e intuitiva interfaz. Su uso es cómodo y predictivo.*

Agregar nuevos medios

Nuestra aplicación ya funciona, solo nos resta buscar nuevos streaming de radios en la web, y grabar los mismos en el archivo de texto. Dentro de **btnAgregar_Click()** agregamos el siguiente código:

```
Dim objStreamReader As StreamWriter, objSR As StreamReader
Dim strLine As String, strNuevaEmisora As String
Dim msg = “Ingrese la nueva emisora con el siguiente formato: Nueva
emisora!http://streaming.nuevaemisora.com/”
    strNuevaEmisora = InputBox(msg, “Nueva Emisora de radio”)
If Trim(strNuevaEmisora) <> “” Then
    objSR = New StreamReader(pathArchivo)
    strLine = objSR.ReadToEnd()
    objSR.Close()
    objStreamReader = New StreamWriter(pathArchivo)
    strLine = strLine & vbCrLf & strNuevaEmisora
    objStreamReader.Write(strLine)
    objStreamReader.Close()
    cargoRadios()
End If
```

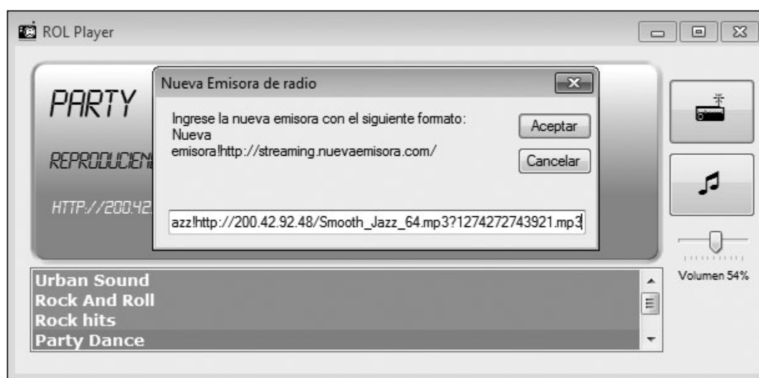


Figura 8. *Ya podemos incorporar nuevas emisoras a nuestra app.*

Ya tenemos listo nuestro propio reproductor de streaming personal. Nada nos queda por envidiar a las grandes aplicaciones como lo son **Windows Media Player**, el **Gadget de Radio**, **iTunes** ó **Songbird**. Para mejorar el mismo podemos estudiar las principales funciones de los recién mencionados y así potenciar el nuestro, evaluando los alcances de estas posibles modificaciones. También recordemos que con algunas líneas de código más podremos transformar nuestro reproductor de radio en un reproductor MP3. Gracias a las herramientas de Windows, vimos que con pocas líneas de código, algunas ideas buenas y mucho entusiasmo al encarar nuestros proyectos, lograremos desarrollar aplicaciones prácticas en tiempo récord

Desarrollo para Windows Phone 7

Realizaremos una introducción a las herramientas propuestas por Microsoft para el desarrollo de Mobile Apps para su nuevo sistema operativo Windows Phone 7. Conoceremos un poco la historia de la llegada de Microsoft al mundo de la telefonía móvil, lo nuevo propuesto a través de Windows Phone y el alcance de las herramientas de desarrollo integradas a la suite Visual Studio 2010.

INTRODUCCIÓN

Los teléfonos celulares ya se han instalado en nuestra vida. Desde 2005 hasta ahora, el mercado de **smartphones** o teléfonos celulares con capacidades de cómputo no ha parado de crecer. Cientos de equipos han invadido el mercado, entre ellos, reconocidas marcas como **Nokia**, **Sony Ericsson** y **Blackberry**, que han hecho un gran trabajo con sus teléfonos celulares al permitir, desde ellos, un cómodo acceso a Internet, edición de documentos ofimáticos, y manejo de correo electrónico y mensajería instantánea.

Apple llegó un poco más tarde a este nicho con su conocido **iPhone**, marcando, como casi siempre, una **re-revolución** de todo lo conocido hasta el momento. Microsoft, por su parte, en 2003 decidió extender su plataforma **Windows CE**, existente desde finales de la década del 90, al mercado de la telefonía celular.

En un principio, el sistema operativo mantuvo su nombre para luego cambiar por **Windows Mobile**. Si bien todo arrancó en 2003, y dicha versión de Visual Studio ya integró soporte para desarrollo de aplicaciones Windows Mobile, recién en mayo de 2005 la empresa pudo integrar una versión estable y aceptable por el público de su plataforma **Pocket PC**.

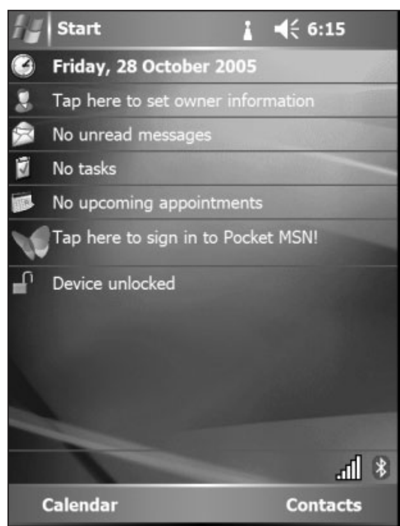


***Figura 1. Pantalla inicial** de **Windows Mobile 5**, el primer sistema operativo para celulares de Microsoft masivamente aceptado por el "mundo mobile".*

.NET Compact Framework

Acompañando el lanzamiento de su nuevo sistema operativo para plataformas móviles, se creó el **.NET Compact Framework**, similar al instalado en las máquinas de escritorio para desarrollar desde la plataforma **Visual Studio**, destinado a cualquier versión existente de Windows CE.

Esto abarcó el desarrollo para **teléfonos móviles**, **handhelds**, **Set Top Boxes**, y lo que fue la segunda generación de computadoras ultraportátiles, antes del lanzamiento de las **netbooks**. El .NET Compact Framework, de esta manera, permitía utilizar algunas de las bibliotecas de clases existentes en el framework .NET, y también las adaptadas para este tipo de dispositivos.
Las primeras plataformas del .NET Compact Framework tuvieron sus desarrollos a partir de **Visual Studio .NET 2003**, pasando luego por **VS 2005** y **VS 2008**. Dichos desarrollos podían llevarse a cabo en el lenguaje **Visual Basic .NET** y en **C#**.
Al igual que las aplicaciones para computadoras de escritorio o portátiles, todos los desarrollos para smartphones ó Pocket PC se ejecutan bajo el compilador **JIT**, diseñado especialmente para este tipo de dispositivos.

LO NUEVO: WINDOWS PHONE 7

Con el incansable crecimiento del mercado de **smartphones**, Microsoft replanteó la estrategia hasta el momento llevada a cabo con su sistema operativo para celulares, y decidió reprogramarlo casi desde cero. Así es como Windows Mobile quedaría para siempre congelado en la versión **6.5.5**, para darle paso a un nuevo sistema operativo: **Windows Phone** 7.
En él se incluyeron las modificaciones necesarias para responder a los requerimientos de los usuarios de poder contar con un equipo que tenga **soporte multitáctil**, mejor respuesta en el uso de **pantallas touch**, y una interfaz renovada y más ágil que la brindada hasta el momento por Windows Mobile.
Así nació Windows Phone 7, y junto con él, una nueva suite de desarrollo de aplicaciones, que dejó de lado por completo el soporte para Windows Mobile 6.xx. Hasta último momento, muchos desarrolladores siguieron los avances del **SDK** para Windows Phone, pero hasta su lanzamiento oficial, Microsoft no reveló que se incluiría en él un nuevo modelo de programación, basado solo en **Silverlight** y **Visual C#**.
A fines de 2010, debido a un gran reclamo de la comunidad de desarrolladores **Visual Basic**, Microsoft lanzó un **Add-On** para que los programadores de la



ÚLTIMOS CAMBIOS SOBRE WP7

Durante marzo de 2010, Microsoft lanzó el SDK Windows Phone 7 para VB 2010. En el momento de armar el índice de este libro, no se tenía agendado el lanzamiento, por lo cual se menciona a Visual C# como el lenguaje para crear aplicaciones para WP7. Podemos descargarlo desde el link: **http://create.msdn.com/en-us/home/getting_started**.

plataforma Visual Basic 2010 pudieran desarrollar aplicaciones para Windows Phone, aunque solamente están soportadas las versiones **Professional**, **Premium** o **Ultimat**e. El enfoque que se da en este apéndice a los desarrollos en Windows Phone 7 solo abarcará la versión de Visual C#. Quienes tengan la versión Professional o superior de Visual Basic 2010 pueden descargar el plugin desde el siguiente link: **www.microsoft.com/downloads/details.aspx?FamilyID=4e97ea70-e479-4c05-814f-639d71690e5d&displayLang=es#Requirements**.



***Figura 2.** Desde el nuevo portal de Windows Phone 7 se pueden descubrir todas las novedades que incluye este sistema operativo para dispositivos móviles: **www.microsoft.com/windowphone**.*

Cómo iniciarse en la programación para smartphones WP7

Junto con la llegada de este nuevo sistema operativo para celulares, Microsoft puso a disposición de los desarrolladores de aplicaciones un conjunto de herramientas para que puedan explorar el nuevo mundo del sistema operativo para celulares, y tengan todo lo necesario para capacitarse en programación para WP7.

Portal App Hub

Desde este nuevo portal, podemos acceder al contenido totalmente gratuito de la comunidad de desarrolladores. Para ingresar, lo hacemos a través de la siguiente dirección: **http://create.msdn.com**.

Channel 9 Video Training

Desde el canal videos de **App Hub** podemos iniciar un curso en video, recorriendo desde el principio todo lo referente al lanzamiento de Windows Phone 7 y sus

características principales, cómo interactuar con servicios web como **Bing Maps** y conocer en detalle los alcances que nos brinda esta plataforma.

Visual Studio 2010 Express para Windows Phone 7

En el portal App Hub se incluye, para desarrolladores, la descarga de una versión de Visual Studio 2010 Express que permite programar para Windows Phone y Xbox 360. Para acceder a él debemos tener una cuenta en **Windows Live**. Si disponemos de una, podemos utilizarla; de lo contrario, tendremos que registrarnos para obtener una, ya sea a través de nuestro correo **Hotmail** o de cualquier otro que utilicemos. Desde el link **http://create.msdn.com/en-us/home/getting_started** podremos descargar la suite de herramientas de Microsoft y acceder también a toda la ayuda disponible en el portal.



***Figura 3.** Desde el apartado Download free tools podemos descargar la versión Express de Visual Studio 2010 para Windows Phone.*

Para poder probar el entorno de desarrollo preparado para el nuevo sistema operativo para celulares de Microsoft, debemos descargar, desde el link mencionado, la edición especial de Visual Studio 2010 Express para plataformas móviles. Luego, iniciamos la instalación correspondiente en nuestro equipo.

Requisitos para ejecutar VS 2010 para Windows Phone

Los requisitos de instalación para esta versión van desde **Windows XP SP3** hasta **Windows Seven Starter** o superior, un mínimo de 2 GB de memoria **RAM** y un procesador de **doble núcleo**, aunque se puede ejecutar también, con algo de paciencia, en un **Intel Atom D410** o superior.

EL IDE DE VS 2010 EXPRESS

Como hemos recorrido el IDE de Visual Basic 2010, nos encontraremos muy familiarizados con el entorno de VS 2010 para Windows Phone. La distribución de **menús**, **barras de herramientas**, **cuadro de herramientas**, entorno de desarrollo, el Visor de objetos y el **cuadro de propiedades** es similar a la de cualquier otro lenguaje de Visual Studio.
Las sentencias de programación sí cambian, dado que aquí debemos programar en el lenguaje **C#** en vez de hacerlo en Visual Basic. Quienes hayan incursionado en el lenguaje **Java** o en alguna variante del lenguaje **C** o **Visual C++** encontrarán mucha similitud entre **Visual C#** y ellos, con el plus de productividad que brinda un IDE visual en cuanto a construcción de interfaces.

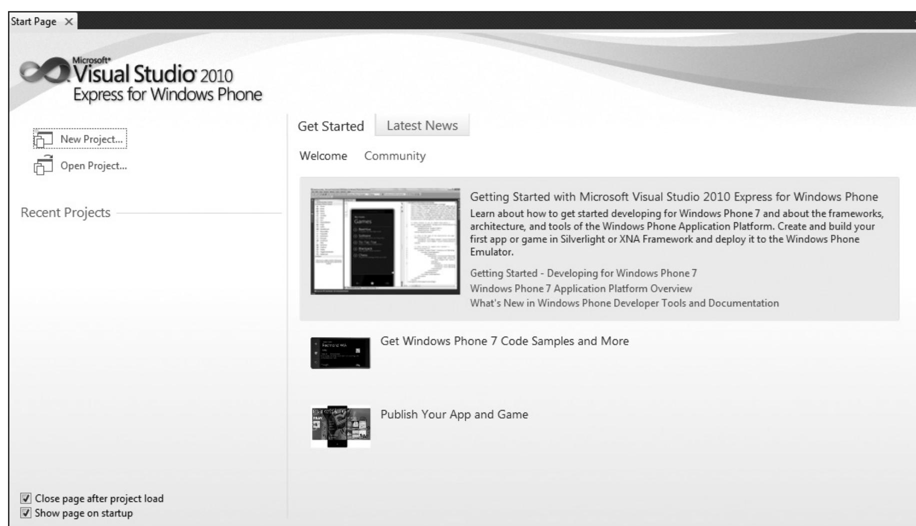


Figura 4. *El IDE de Visual Studio 2010 para Windows Phone contiene las mismas características que el IDE de Visual Basic 2010.*

Vamos a iniciar un proyecto para comprender mejor la nueva área de construcción de interfaz. Para esto, nos dirigimos al menú **File/New Project**, y de las opciones que



DESARROLLO PARA CELULARES

La llegada de los teléfonos celulares inteligentes, o smartphones, trajo una diversidad de sistemas operativos móviles impensada para muchos. Por suerte, la industria de la telefonía móvil entendió que **Java** en su momento era **"el lenguaje"** para desarrollar aplicaciones móviles, y prepararó los SO para que fuera fácil captar desarrolladores.

aparecen, seleccionamos **Windows Phone Application**. Como nombre le ponemos **ConversorDeGrados** y presionamos el botón **OK**.

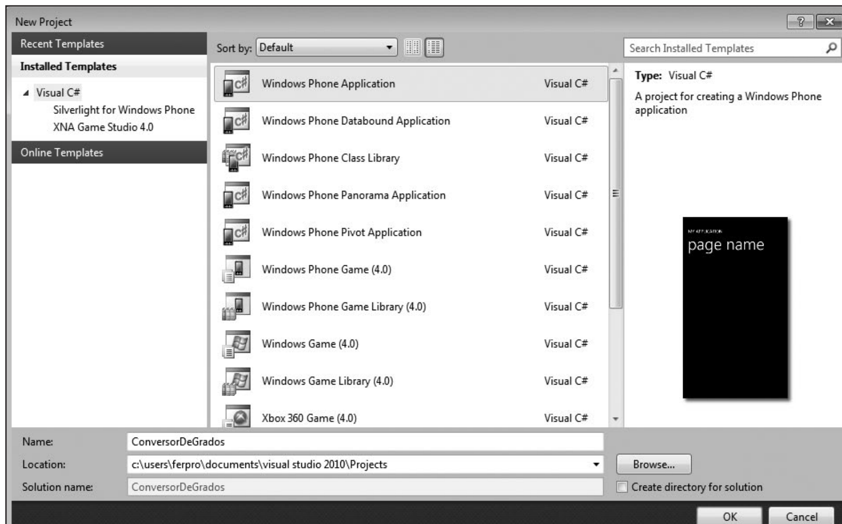


***Figura 5.** De todas las opciones de proyecto que podemos crear con este **IDE**, seleccionamos la primera: **Windows Phone Application**.*

El clásico **Form1** de Visual Basic 2010 es reemplazado por un área titulada **MainPage.xaml**, que simula un teléfono táctil. El teléfono virtual, dependiendo de la configuración de nuestro display, tal vez se vea demasiado grande. Para nuestra comodidad se incluye en el extremo superior izquierdo de dicha pestaña un regulador de **Zoom** que permitirá ver cómodamente al emulador.

Podemos ajustar su tamaño hasta que este quede cómodo en el área de trabajo del monitor de la computadora. Sobre el lateral izquierdo del área central del IDE encontramos una página con código **XAML**.

Como en este ejemplo no vamos a trabajar sobre este código, podemos ocultar este panel desde el extremo inferior del marco central divisor de paneles, presionando la opción **Collapse Pane**. Sobre el lateral derecho podemos ver el **Cuadro de propiedades**, donde, tal como lo hicimos en los distintos ejemplos brindados en



NACIMIENTO DE WINDOWS MOBILE

Microsoft ingresó en el mundo de los teléfonos inteligentes con el mercado bastante avanzado. Sin embargo, ya tenía experiencia en dispositivos portátiles gracias a su versión reducida de **Windows CE**. Esto le allanó el terreno para ingresar en el campo de la telefonía móvil, aunque no fue tan fácil ganar la aceptación de los usuarios.

este libro, es posible ajustar las propiedades principales de los controles que distribuiremos en nuestra **Mobile App**.



***Figura 6.** El IDE de Visual Studio. El clásico Form1 es reemplazado por un área de trabajo similar a un teléfono celular **touchscreen**.*

Programación de aplicaciones para WP7

El proyecto que abordaremos como ejemplo consistirá en realizar un **Conversor de temperatura** para los teléfonos Windows Phone 7. En él se incluirá la conversión posible entre grados **Celsius** y **Fahrenheit**. Como primera medida, debemos agregar los componentes al área de trabajo.

Una vez creada la interfaz visual de la aplicación, procedemos a agregar el código necesario para que esta funcione. Toda aplicación Windows Phone 7 dispone, en su margen superior, de dos componentes **TextBlock**: el primero indica el nombre de la aplicación, y el segundo, el título de la página que estamos visualizando.

Sobre el resto del área de la aplicación distribuimos los controles detallados en la **Tabla 1**. Los dos primeros componentes allí listados son los que aparecen por defecto en nuestra área de trabajo, propios del desarrollo.

TIPO DE CONTROL			
CONTROL	NOMBRE	PROPIEDAD	VALOR
TextBlock	ApplicationTitle	Text	Conversor de Grados C° <> F°
TextBlock	PageTitle	Text	CF Converse
TextBlock	TextBlock1	Text	Ingrese la temperatura:

TIPO DE CONTROL			
CONTROL	NOMBRE	PROPIEDAD	VALOR
TextBox	TextBox1	Text	
TextBlock	TextBlock2	Text	Transformar a:
RadioButton	RadioButton1	Content	Celsius
RadioButton	RadioButton2	Content	Fahrenheit
Button	Button1	Content	Convertir
TextBlock	TextBlock3	Text	Resultado:
TextBlock	TextBlock4	Text	Value
Button	Button2	Content	Reset

Tabla 1. *Los controles para agregar están indicados en el orden correspondiente en que debemos añadirlos a nuestro proyecto.*

Si queremos, podemos especificar un color y un tamaño de fuente determinado para cada control **TextBlock**, tendría que ser comando que permitan distinguir bien cada sección componente de nuestra aplicación. En la **Tabla 2** podemos apreciar las propiedades utilizadas para tal fin en este proyecto de ejemplo.

CONTROL	PROPIEDAD	VALOR
TextBlock1	Foreground	#FFEF5050
TextBox1	Background	#BFFFFFFF
TextBlock2	TextBlock2	#FFEF5050
RadioButton1	Font	Segoe WP
	FontSize	22.667
RadioButton2	Font	Segoe WP
	FontSize	22.667
Button1	Font	Segoe WP Semibold
	FontSize	25.333
TextBlock3	Font	Segoe WP Semibold
	FontSize	25.333
TextBlock4	Foreground	#FFEF5050
Button2	Font	Segoe WP Semibold
	FontSize	25.333

Tabla 2. *Algunos valores de las propiedades de los controles fueron cambiados para que la aplicación tenga un aspecto más cómodo a la vista.*

Recuerden la convención de nombres de la que hablamos al iniciar nuestros proyectos. Si desean aplicar la misma para cada control, tomen nota de los nombres usados y elaboren en papel el reemplazo para cada uno de ellos. Una vez agregados todos los controles que conforman el proyecto y ajustadas sus propiedades principales, el diseño debe quedar distribuido similar a lo que se observa en la **Figura 7**.



***Figura 7.** Así queda la distribución de los controles en el área de trabajo de nuestra aplicación Windows Phone 7.*

La finalidad de nuestra aplicación es ingresar un número, que equivale a una escala de temperatura que deseamos transformar. Si en nuestro país utilizamos la escala Celsius y solemos viajar a otras partes del mundo donde la escala térmica está expresada en Fahrenheit, podremos conocer la equivalencia de la temperatura local con respecto a la nuestra, o viceversa.
En la **Tabla 3** veremos el cálculo matemático necesario para conocer de qué manera transformamos una unidad térmica en otra.

TRANSFORMACIÓN DE CELSIUS A FAHRENHEIT	FÓRMULA MATEMÁTICA	RESULTADO
Temperatura 78 C°	(78 – 32) * 5 (/ 9)	25.5 C°
TRANSFORMACIÓN DE FAHRENHEIT A CELSIUS	**FÓRMULA MATEMÁTICA**	**RESULTADO**
Temperatura 25.5 C°	(25.5 * 9) / 5 (+ 32)	78 C°

***Tabla 3.** Fórmula de conversión de temperaturas entre grados **Fahrenheit** y **Celsius**.*

A continuación, agregaremos el código correspondiente para que la app funcione. Para hacerlo, debemos tener en cuenta que las sentencias por utilizar están escritas en C#. El estilo de programación de C# es similar al de Visual Basic y es por eso que encontraremos muchas coincidencias entre estos dos lenguajes.
El código para que nuestra aplicación funcione se centrará, exclusivamente, en los dos botones que agregamos: `Button2` oficiará como Reset de los valores mostrados en la pantalla, y `Button1` concentrará el cálculo matemático para

obtener el resultado de la conversión, dependiendo de si elegimos convertir la temperatura a grados Celsius o a grados Fahrenheit.
Agregamos en primera instancia el código de **Button2**. Para esto, al igual que en los desarrollos llevados a cabo durante todo el libro, realizamos doble clic sobre el control **Button2**. A continuación, entre las llaves de inicio y cierre del evento **button2_click()** escribimos el siguiente código:

```
private void button2_Click(object sender, RoutedEventArgs e)
{
    textBox1.Text = “”;
    textBlock4.Text = “Value”;
    textBox1.Focus();
}
```

Este código limpia el contenido de **TextBox1**, aplica el valor **Value** a **TextBlock4** y devuelve el foco a **TextBox1** para poder ingresar otro valor para convertir. Recordemos finalizar cada línea de código escrita con un **punto y coma**. Esto indica el final de línea en todos los lenguajes de programación que son variantes de **C**. Las llaves de inicio y cierre se utilizan en los eventos de cada control, cuando creamos una función o procedimiento y al utilizar las sentencias e instrucciones del lenguaje. Su función es indicar el inicio y el cierre de cada algoritmo escrito y estructurar el código para facilitar la lectura. Luego nos queda agregar el código correspondiente al evento **click()** de **Button1**:

```
private void button1_Click(object sender, RoutedEventArgs e)
    {
     double temp;
     double subt;
     double total;
     temp = System.Convert.ToDouble(textBox1.Text);
     if (radioButton1.IsChecked == true)
//Si tildaron Celsius
     {
         subt = temp - 32;
         total = subt * 5;
         total = total / 9;
         textBlock4.Text = Convert.ToString(total);
     }
     if (radioButton2.IsChecked == true)
```

```
//Si tildaron Fahrenheit
               {
                    subt = temp * 9 / 5;
                    total = subt + 32;
                    textBlock4.Text = Convert.ToString(total);
               }
              }
```

En primer lugar, declaramos tres variables: **temp**, **subt** y **total** del tipo **Double**. En ellas almacenaremos los valores con los cuales se llevarán a cabo las operaciones matemáticas para llegar al resultado deseado.
Luego utilizamos **System.Convert.ToDouble()** para transformar el valor ingresado en **TextBox1.Text** al tipo de dato **Double**. Por último, empleamos dos instrucciones **if** para verificar qué **RadioButton** ha sido chequeado. Con el estado de este último control podremos conocer qué tipo de operación debemos aplicar para convertir el valor ingresado a los grados de destino elegidos.
Una vez realizada la conversión correspondiente, ésta quedará almacenada en la variable total. Luego, cargamos el contenido de la variable en **TextBlock4.Text** haciendo uso de la función **Convert.ToString()**.

```
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
            {
             double temp;
             double subt;
             double total;
             temp = System.Convert.ToDouble(textBox1.Text);
             if (radioButton1.IsChecked == true) //Si tildaron Celsius
             {
                 subt = temp - 32;
                 total = subt * 5;
                 total = total / 9;
                 textBlock4.Text = Convert.ToString(total);
             }
             if (radioButton2.IsChecked == true) //Si tildaron Fahrenheit
             {
                 subt = temp * 9 / 5;
                 total = subt + 32;
                 textBlock4.Text = Convert.ToString(total);
             }
            }

        private void button2_Click(object sender, RoutedEventArgs e)
        {
```

Figura 8. *Con tan solo algunas líneas de código, nuestro proyecto WP7 ya funcionará sin problemas.*

Ya hemos finalizado el código que hará que nuestra aplicación sea funcional. Guardemos todo lo realizado hasta el momento para no perder nada del trabajo y procedamos a probar el resultado. Al igual que en Visual Basic 2010, nuestros desarrollos para **Windows Phone 7** pueden ejecutarse presionando **F5**, desde la barra de herramientas del IDE de desarrollo o desde el menú **Debug/Start Debugging**.

Iniciamos el proyecto desde alguna de estas opciones mencionadas. Este se cargará en el entorno de prueba de Windows Phone. Al finalizar la carga de la aplicación, esta simulará todo ingreso de caracteres presentando el teclado virtual de un teléfono celular con pantalla touch. Este aparece de manera automática en la pantalla cuando una caja de texto recibe el foco. Para simular el desplazamiento y las selecciones de opciones del programa sí debemos utilizar el mouse.



***Figura 9.** Nuestra aplicación corriendo en el entorno de pruebas de Windows Phone 7.*

Ya tenemos el entorno iniciado para probarla. Para ello, hacemos un clic con el mouse sobre la caja de texto para ingresar una temperatura. A continuación veremos que aparecerá automáticamente el teclado virtual en pantalla.
Presionamos sobre la tecla inferior izquierda **&123** para habilitar el **numPad**, e ingresamos con un clic la temperatura que deseamos transformar.



***Figura 10.** El teclado virtual que aparecerá en nuestras apps cada vez que debamos ingresar un texto.*

Por último, vamos a ingresar un valor, expresando en grados Fahrenheit, para luego transformar el mismo a grados Celsius. Colocamos **86** utilizando el teclado virtual en pantalla. Luego, seleccionamos **Celsius** desde el **OptionButton** correspondiente y, por, último presionemos el botón **Convertir**. Obtendremos el resultado en pantalla tal como lo muestra la **Figura 11**.



***Figura 11.** Ya con el resultado en pantalla, probemos **resetear** los valores e ingresar, por ejemplo, 30 para obtener su equivalente en Fahrenheit.*

Gracias a la facilidad del IDE de .NET para llevar a cabo aplicaciones completamente funcionales para Windows Phone 7, nuestras ideas para explotar al máximo el mercado de software para telefonía celular podrán ser llevadas a cabo sin ningún inconveniente bajo este renovado IDE de desarrollo.

No dejen de explorar el portal **App Hub** de **Microsoft**, donde ofrecen tutoriales en video y cientos de aplicaciones de ejemplo desarrolladas para conocer al máximo las prestaciones que VS 2010 pone a nuestro alcance.

Diferencias entre VB.NET y C#

Hemos dejado para el final del libro una comparativa de similitudes y diferencias entre el lenguaje Visual Basic .NET y un nuevo lenguaje llegado hace tan solo una década: Visual C#. Repasaremos muy brevemente la historia de este lenguaje y veremos las ventajas que ofrece C# sobre Visual Basic .NET. También estudiaremos su sintaxis y ejercitaremos un poco cómo aplicar las sentencias aprendidas en Visual Basic .NET directamente en Visual C#.

C#, UN LENGUAJE JOVEN Y PODEROSO

Desde la llegada de **.NET**, Microsoft ha facilitado mucho a los desarrolladores el hecho de tener que distribuir las librerías necesarias para que un proyecto creado con alguno de sus lenguajes de programación funcionase bien.

Como mencionamos al principio del libro, el **framework .NET** agrupa todas las librerías e intérpretes necesarios para que los lenguajes de desarrollo de Microsoft funcionen sin ningún inconveniente. **Visual Basic** se ha vuelto el lenguaje más popular y utilizado en todo el mundo en las últimas dos décadas, gracias a la facilidad de aprendizaje que podemos encontrar en él.

Visual Fox Pro dejó hace unos años de integrar la suite de Microsoft, **Visual C++** sigue formando parte del equipo, y desde hace casi una década, **Visual C#** se hizo un lugar también dentro de la suite de desarrollo ofrecida por el gigante de **Redmond**, y con los años fue ganando la empatía de la gente, casi emparejándose con Visual Basic en cuanto a popularidad.

También se pudo apreciar en el mercado IT que, desde hace unos cuatro años, el skill de programadores para con este lenguaje comenzó a ser muy solicitado. **C#** (# se pronuncia **Sharp**) es un lenguaje del tipo **POO** (**Programación Orientada a Objetos**) en el cual Microsoft trabajó desde 1999 para la plataforma .NET, y que pasó a pertenecer a la plataforma Visual Studio .NET desde su primera versión.

En 2001 se certificó al lenguaje como un estándar **ECMA** e **ISO**. Su sintaxis deriva de **C/C++**, aunque fue preparado para utilizar el modelo de objetos del framework .NET. C# nació de la idea de dibujar dos signos **+** por sobre los signos de **C++**, con lo que Microsoft quiso dar a entender que C# era una mejora o evolución del lenguaje base C++, el cual, a su vez, en su momento indicó a través del signo **+** que era una evolución de **C**.

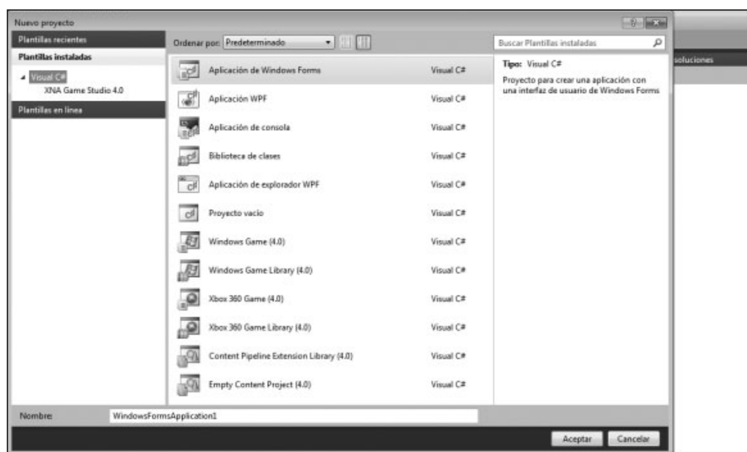


Figura 1. *Visual C#, desde su versión Express, permite crear una variedad de aplicaciones, desde **Windows Forms**, **XAML** y **ASP.NET**, hasta **Apps** y **juegos** para **WP7** y para **X-Box 360**.*

Ventajas que ofrece C#

En muy poco tiempo, C# ha logrado ganarse los primeros puestos entre los lenguajes de programación elegidos para el desarrollo de soluciones de baja, mediana y gran escala. Desde una simple aplicación de escritorio que permita escuchar música, pasando por un procesador de textos, un sistema de base de datos de escritorio o Web, desde la versión 2010 del lenguaje, C# es el elegido principal para crear aplicaciones para Windows Phone 7, como vimos en el **Apéndice B** de este libro, y también para el desarrollo de aplicaciones y juegos para **XNA**, la plataforma Microsoft de juegos para **Xbox 360** y PC.

C# logró combinar el poder de C++, la productividad en desarrollo visual propio de Visual Basic, y algunas particularidades que tuvo el lenguaje **Delphi** en su momento de popularidad, hace alrededor de diez años.

Una de las ventajas que ofrece Visual C# por sobre VB.NET es el soporte completo para poder desarrollar aplicaciones Windows Phone y juegos para la plataforma móvil y para Xbox 360. Incluso en el desarrollo para Windows Phone se pueden usar servicios especiales, como los mapas Bing, entre otros.

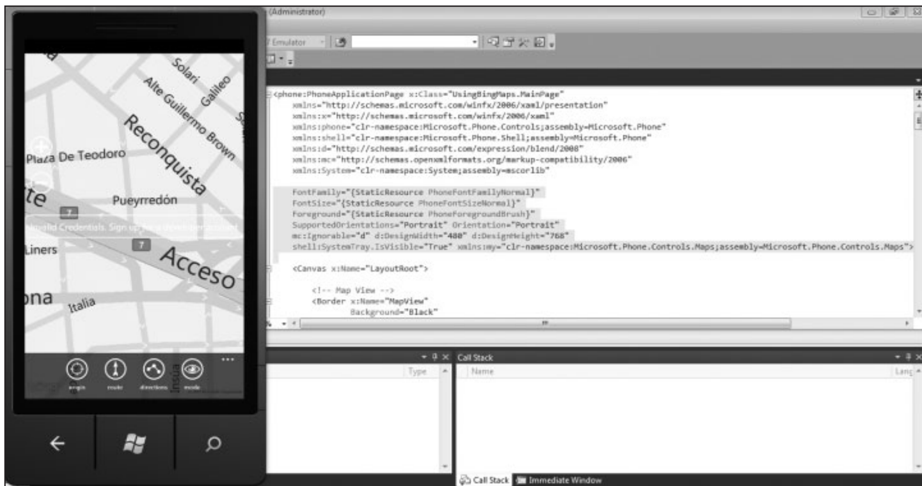


Figura 2. *El servicio* ***Bing Maps*** *brinda soporte de geoposicionamiento y APIs para que pueda ser utilizado desde aplicaciones Windows Forms y, también, en* ***Windows Phone 7****.*

ORÍGENES DE C#

C# es un lenguaje que nació de la mano de Microsoft, en el que se mezcla mucho del antiguo **Visual J++**, una implementación de la empresa para un lenguaje propietario que corriera una versión mejorada de Java. Con el lanzamiento de la primera versión de .NET, llegó **J#**, que luego desapareció por completo para dejarle lugar a **C Sharp**.

SIMILITUDES Y DIFERENCIAS ENTRE VB.NET Y C#

Visual C# se diferencia de Visual Basic por corresponder a la sintaxis propia del lenguaje C. **Java** y **PHP** también derivan de este lenguaje, con lo cual cualquier persona que desee aprender C# y que ya conozca alguno de los otros lenguajes, tendrá una curva de aprendizaje muy rápida.
El lenguaje C# también permite crear documentación **XML** directamente desde el código fuente. **Refactoring** y **Snippets automáticos** lo diferencian de Visual Basic. Los operadores de comparación también tienen algunas diferencias entre un lenguaje y otro. En la **Tabla 1** podemos ver en detalle una clara comparativa entre **Visual Basic 2010** y **Visual C# 2010**.

OPERADOR	VISUAL BASIC 2010	VISUAL C# 2010
Suma	+	
Resta	-	
Multiplicación	*	
División	/	
Resto de un entero	%	
Incremento numérico en una variable	i = i + 1	i++
Decremento numérico en una variable	j = j - 1	j–
Comparativa de igualación	A = B	A == B
Comparativa si es distinto que	A <> B	A != B
Mayor que	A > B	
Menor que	A < B	
Mayor o igual que	A >= B	
Menor o igual que	A <= B	A <= B
Operador lógico AND	A = B and C = D	A = B & C = D
Operador lógico ORº	A = B or C = D	A = B | C = D
Operador lógico OR de cortocircuito	NOT A = B	A ^ B
Operador lógico AND de cortocircuito	A = B AND NOT C = D	A = B && C = D
Operador lógico NOT	A NOT B	A ! B

***Tabla 1.** Similitudes y diferencias en los operadores en VB.NET y C#.*



ALCANCES DEL LENGUAJE

Si bien C# es parte del framework y plataforma .NET, debemos reconocerlo también como un lenguaje de programación independiente, ya que existe un compilador basado en el framework DotGNU, conocido como Mono, que permite generar programas tanto para Win32, como para UNIX y Linux. Más información en **www.mono-project.com**.

La declaración de variables siempre es obligatoria antes de su uso, al igual que en todo lenguaje orientado a objetos, y al momento de declararla, ya se le puede asignar un valor. Los tipos de datos en C# son iguales a los utilizados en Visual Basic. Veamos a continuación un ejemplo:

```
//Declaración de una variable string


String strTituloVentana;


//Y asignándole un valor en el momento de su declaración sería:

String strTituloVentana = “ABM Conductores”;
```

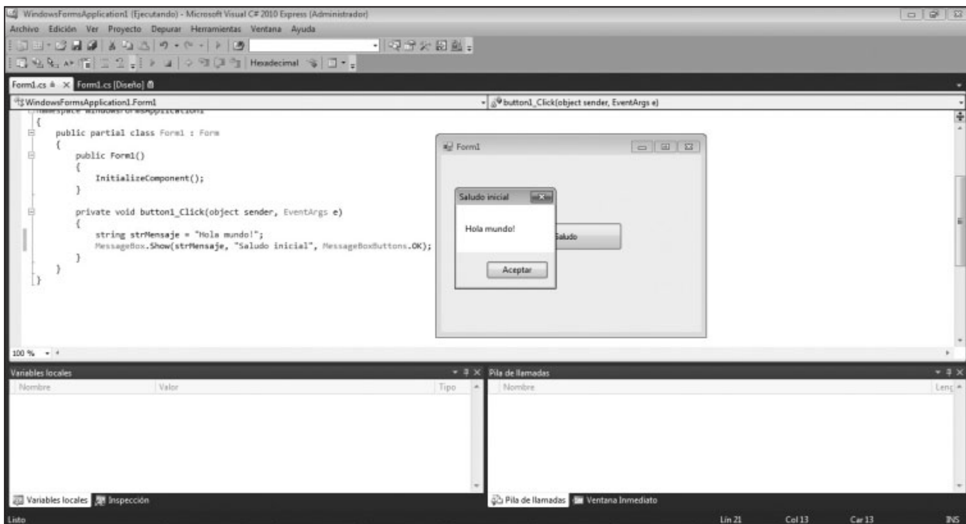


Figura 3. *El clásico “Hola mundo”, aplicando la creación de una variable junto con la asignación de su valor, realizado en Visual C#.*



SIMILITUD EN INSTRUCCIONES

Las instrucciones **if-else**, **Switch**, **For**, **While**, **Do While**, **Foreach**, **break**, **Continue** y **Return** se utilizan de la misma manera en C# que otros lenguajes similares, como C, C++ y Java. **Goto** también se puede usar, aunque la polémica con respecto a ella es muy grande, dado que únicamente solía emplearse en lenguajes no orientados a objetos.

Todo código dentro de eventos, funciones y procedimientos en una aplicación C# lleva llaves (**{ }**) que indican su inicio y fin, como así también se aplican en el inicio y fin de una instrucción. Los comentarios que se deben realizar dentro de la aplicación o en aquellos casos en que hay que comentar un código para que no se ejecute, se efectúan con una doble contrabarra (**//**) al inicio.
En caso de que debamos comentar varias líneas de código, se inicia el comentario con una contrabarra seguida de un asterisco (**/***), y al final se cierra con un asterisco seguido de una contrabarra (***/**), tal como vemos en la **Figura 4**.

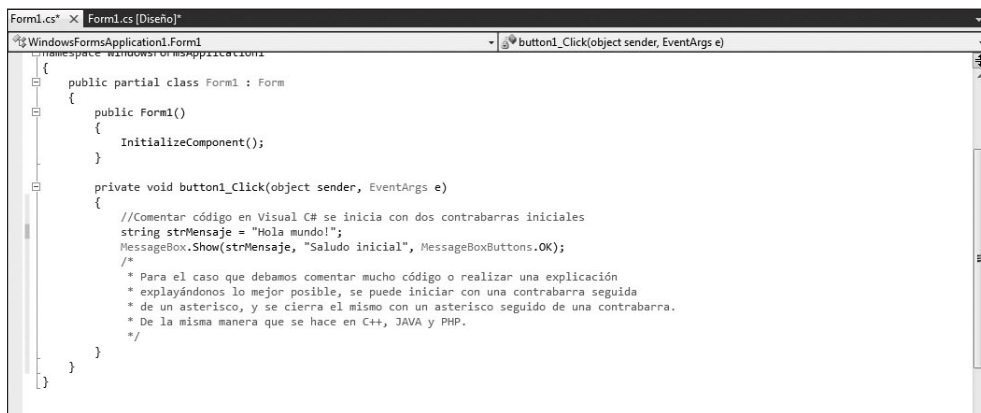


***Figura 4.** Los comentarios de código en C# también poseen similitudes con C, C++ y Java.*

Todos los **namespaces** utilizados en Visual Basic 2010 son visibles en Visual C#, con lo cual, habiendo aprendido sobre ellos, comenzar a programar en C# nos hará sentir más cómodos, ya que tendremos el principal conocimiento de namespaces y sabremos cómo aplicarlos en el lenguaje.

SINTAXIS DEL LENGUAJE C#

Veamos a continuación cómo se aplican las instrucciones más comunes dentro de C#. Para comprender claramente las mismas, haremos una comparación entre una instrucción en Visual Basic y su equivalente en este lenguaje. La instrucción **if** en Visual Basic se utiliza de la siguiente manera:

```
'Visual Basic

If coneccionRealizada Then
    MessageBox.Show("Se ha conectado a la base de datos.")
```

```
End If


// Visual C#

If (conexionRealizada == true)
     {
     MessageBox.Show("Se ha conectado a la base de datos.");
}
```

Veamos ahora cómo realizar una **operación aritmética**:

```
'Visual Basic

Public Sub operación()
Dim n as integer = 35
Dim o as integer = 38
Dim result as integer = n * o
End sub

// Visual C#

Public void Main()
{
int n = 35;
int o = 38;
int result = n * o;
}
```

Para agregar un namespace a nuestro proyecto:

```
' Visual Basic

Imports System.IO

// Visual C#
Using System.IO;
```

```
  . . .
   private void button1_Click(object sender, EventArgs e)
   {
       StreamReader srder;
       srder = new StreamReader();
       string strLinea;
       string rutaArchivo = “C:\\cluf.txt”;
       strLinea = srder.ReadLine();


 ... // más código

   }
```

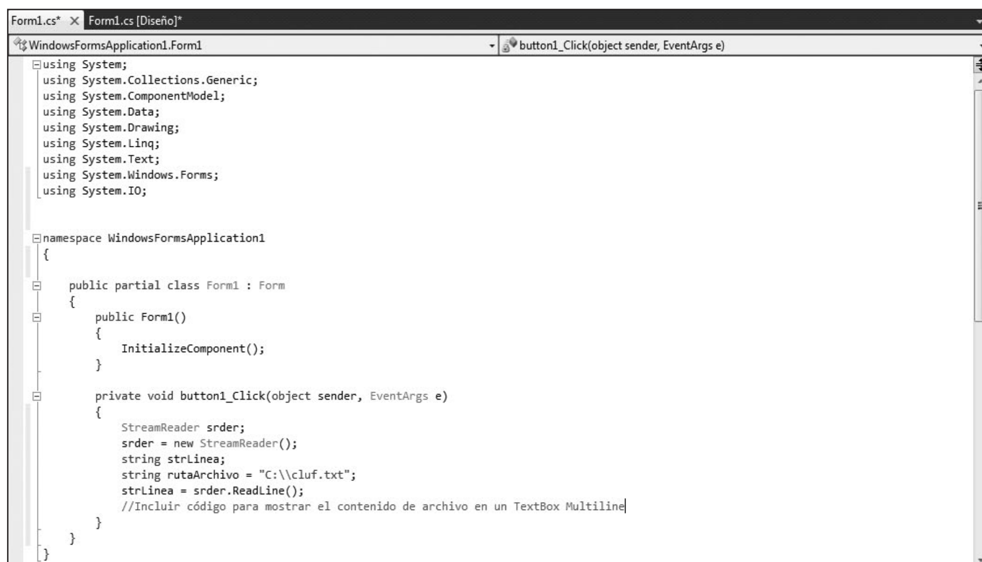


Figura 5. *Especificar un **Path** en C# requiere incluir una doble barra en la ruta, dado que los **strings** utilizan **caracteres de escape**.*



MATRICES EN C#

C# también permite el uso de matrices. La sintaxis de uso correspondiente es **tipo[] nombre_de_matriz = new tipo[nuevotamaño];**. Y si deseamos utilizar una matriz bidimensional, debemos optar por el siguiente formato: **tipo[,] nombre_de_matriz = new tipo[fila_matriz, columna_matriz];** a diferencia de C++, que usa **nombre_de_matriz[fila] [columna]**.

Para poder realizar conexiones a bases de datos, disponemos del mismo asistente de conexión que en Visual Basic, con lo cual no nos será complicado tener que mostrar en una grilla los datos almacenados en una tabla de Access, SQL Server o cualquier otro motor de base de datos.

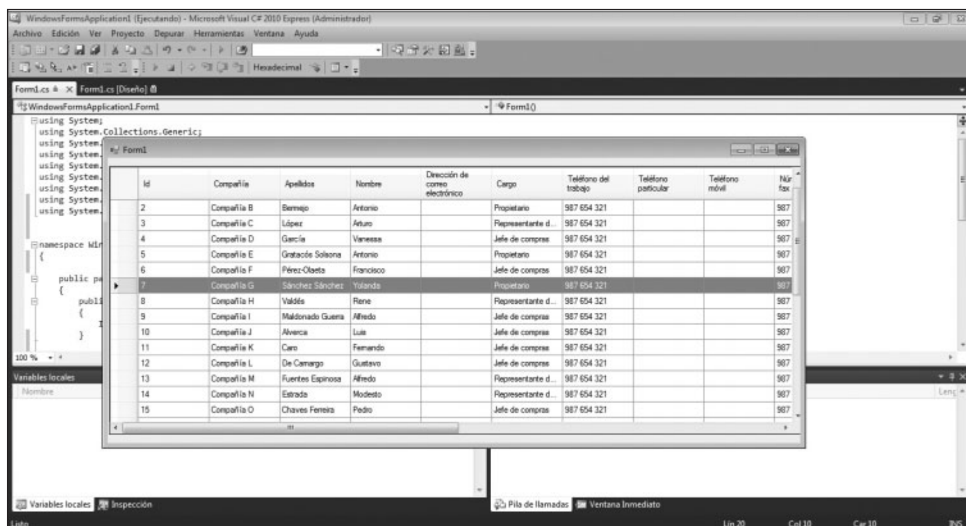


Figura 6. *Utilizando el Asistente de conexión, se estableció el enlace con la base de datos* ***Northwind 2007*** *y se está visualizando la tabla* **Clientes** *en un* **DataGridView***.*

También podremos utilizar la declaración de los objetos correspondientes a bases de datos y tablas, tal como lo hacemos en Visual Basic. Las instrucciones son las mismas, solo cambia la manera de declarar las variables y el punto y coma que debemos incluir al final de cada instrucción escrita.

```
// Debemos incluir en principio las referencias a los namespaces

using System;
using System.Data;
using System.Data.SqlClient;

// Luego se declara el ConecctionString dentro de una clase

class Class1 { static void Main(string[] args)
{
string sConnectionString = “User ID=<UID>;Initial Catalog=pubs;Data
Source=(local)”; }
}
```

Luego se llama al método **Open()** del objeto **connection**, el cual nos permitirá fácilmente establecer la conexión a la base de datos:

```
SqlConnection objConn = new SqlConnection(sConnectionString);
objConn.Open();
```

A continuación, hay que el objeto **sqlCommand** u **OleDBCommand**, dependiendo del tipo de conexión elegido. Veamos una instrucción **INSERT** de SQL:

```
string sSQL = "INSERT INTO Empleados " + "( id, name, lname, jid,
hire_date) " + "VALUES ('1234', 'Fernando', 'Luna', 'W', '1996-07-14')";
SqlCommand objCmd = new SqlCommand(sSQL,objConn);
```

Y, por último, se llama al comando **ExecuteNonQuery()** para procesar el **INSERT** creado en el paso anterior sin esperar un resultado a mostrar:

```
objCmd.ExecuteNonQuery();
```

Como vimos hasta aquí, conectarse a una base de datos, cualquiera sea su tipo, y realizar operaciones sobre ella, no es para nada complicado en C#; de hecho, es muy similar a lo realizado en Visual Basic. Veamos en la **Tabla 2** algunos links recomendados para conocer un poco más el lenguaje C#.

Por último, podemos tener en cuenta también el libro **C#, La guía total del programador**, de esta misma editorial, en donde encontraremos cientos de ejercicios prácticos que nos ayudarán a comprender mejor el lenguaje.

Es altamente recomendable aprender más de un lenguaje de programación, lo que nos favorecerá al momento de buscar empleo o para ofrecernos como programadores. Dada la familiaridad de C# con Visual Basic, y ya que ambos comparten algo en común, que es el framework .NET, es más que lógico que, al aprender un nuevo lenguaje, sabiendo alguno de los dos primeramente, los tiempos de aprendizaje se reducirán de manera considerable. Es posible obtener más información del libro recomendado en el siguiente link: **www.redusers.com/noticias/libro-c-sharp-guia-total-del-programador**.

Diferencias entre VB.NET y C#

Hemos dejado para el final del libro una comparativa de similitudes y diferencias entre el lenguaje Visual Basic .NET y un nuevo lenguaje llegado hace tan solo una década: Visual C#. Repasaremos muy brevemente la historia de este lenguaje y veremos las ventajas que ofrece C# sobre Visual Basic .NET. También estudiaremos su sintaxis y ejercitaremos un poco cómo aplicar las sentencias aprendidas en Visual Basic .NET directamente en Visual C#.

C#, UN LENGUAJE JOVEN Y PODEROSO

Desde la llegada de **.NET**, Microsoft ha facilitado mucho a los desarrolladores el hecho de tener que distribuir las librerías necesarias para que un proyecto creado con alguno de sus lenguajes de programación funcionase bien.
Como mencionamos al principio del libro, el **framework .NET** agrupa todas las librerías e intérpretes necesarios para que los lenguajes de desarrollo de Microsoft funcionen sin ningún inconveniente. **Visual Basic** se ha vuelto el lenguaje más popular y utilizado en todo el mundo en las últimas dos décadas, gracias a la facilidad de aprendizaje que podemos encontrar en él.
Visual Fox Pro dejó hace unos años de integrar la suite de Microsoft, **Visual C++** sigue formando parte del equipo, y desde hace casi una década, **Visual C#** se hizo un lugar también dentro de la suite de desarrollo ofrecida por el gigante de **Redmond**, y con los años fue ganando la empatía de la gente, casi emparejándose con Visual Basic en cuanto a popularidad.
También se pudo apreciar en el mercado IT que, desde hace unos cuatro años, el skill de programadores para con este lenguaje comenzó a ser muy solicitado. **C#** (# se pronuncia **Sharp**) es un lenguaje del tipo **POO** (**Programación Orientada a Objetos**) en el cual Microsoft trabajó desde 1999 para la plataforma .NET, y que pasó a pertenecer a la plataforma Visual Studio .NET desde su primera versión.
En 2001 se certificó al lenguaje como un estándar **ECMA** e **ISO**. Su sintaxis deriva de **C/C++**, aunque fue preparado para utilizar el modelo de objetos del framework .NET. C# nació de la idea de dibujar dos signos **+** por sobre los signos de **C++**, con lo que Microsoft quiso dar a entender que C# era una mejora o evolución del lenguaje base C++, el cual, a su vez, en su momento indicó a través del signo **+** que era una evolución de **C**.

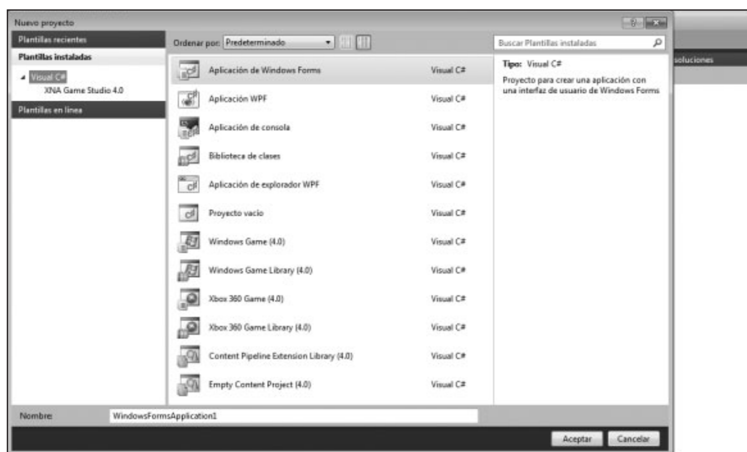


***Figura 1.** Visual C#, desde su versión Express, permite crear una variedad de aplicaciones, desde **Windows Forms**, **XAML** y **ASP.NET**, hasta **Apps** y **juegos** para **WP7** y para **X-Box 360**.*

Ventajas que ofrece C#

En muy poco tiempo, C# ha logrado ganarse los primeros puestos entre los lenguajes de programación elegidos para el desarrollo de soluciones de baja, mediana y gran escala. Desde una simple aplicación de escritorio que permita escuchar música, pasando por un procesador de textos, un sistema de base de datos de escritorio o Web, desde la versión 2010 del lenguaje, C# es el elegido principal para crear aplicaciones para Windows Phone 7, como vimos en el **Apéndice B** de este libro, y también para el desarrollo de aplicaciones y juegos para **XNA**, la plataforma Microsoft de juegos para **Xbox 360** y PC.

C# logró combinar el poder de C++, la productividad en desarrollo visual propio de Visual Basic, y algunas particularidades que tuvo el lenguaje **Delphi** en su momento de popularidad, hace alrededor de diez años.

Una de las ventajas que ofrece Visual C# por sobre VB.NET es el soporte completo para poder desarrollar aplicaciones Windows Phone y juegos para la plataforma móvil y para Xbox 360. Incluso en el desarrollo para Windows Phone se pueden usar servicios especiales, como los mapas Bing, entre otros.

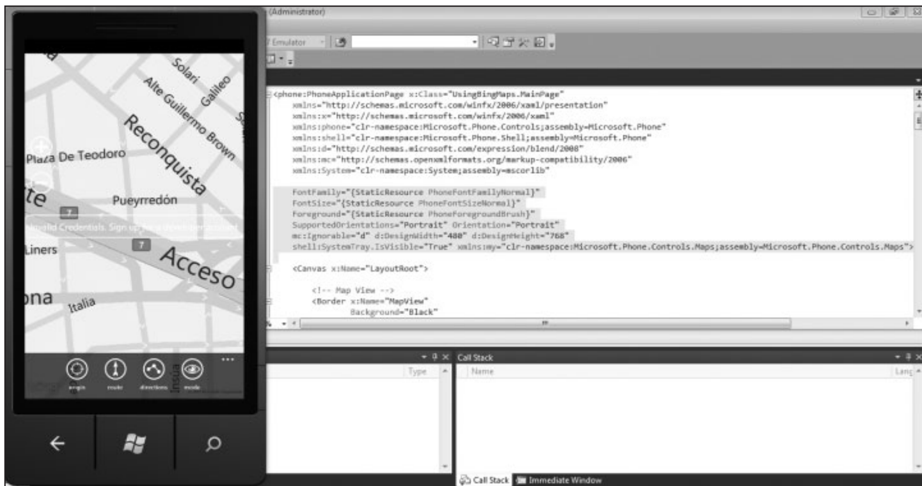


***Figura 2.** El servicio **Bing Maps** brinda soporte de geoposicionamiento y APIs para que pueda ser utilizado desde aplicaciones Windows Forms y, también, en **Windows Phone 7**.*

ORÍGENES DE C#

C# es un lenguaje que nació de la mano de Microsoft, en el que se mezcla mucho del antiguo **Visual J++**, una implementación de la empresa para un lenguaje propietario que corriera una versión mejorada de Java. Con el lanzamiento de la primera versión de .NET, llegó **J#**, que luego desapareció por completo para dejarle lugar a **C Sharp**.

SIMILITUDES Y DIFERENCIAS ENTRE VB.NET Y C#

Visual C# se diferencia de Visual Basic por corresponder a la sintaxis propia del lenguaje C. **Java** y **PHP** también derivan de este lenguaje, con lo cual cualquier persona que desee aprender C# y que ya conozca alguno de los otros lenguajes, tendrá una curva de aprendizaje muy rápida.

El lenguaje C# también permite crear documentación **XML** directamente desde el código fuente. **Refactoring** y **Snippets automáticos** lo diferencian de Visual Basic. Los operadores de comparación también tienen algunas diferencias entre un lenguaje y otro. En la **Tabla 1** podemos ver en detalle una clara comparativa entre **Visual Basic 2010** y **Visual C# 2010**.

OPERADOR	VISUAL BASIC 2010	VISUAL C# 2010
Suma	+	
Resta	-	
Multiplicación	*	
División	/	
Resto de un entero	%	
Incremento numérico en una variable	i = i + 1	i++
Decremento numérico en una variable	j = j - 1	j–
Comparativa de igualación	A = B	A == B
Comparativa si es distinto que	A <> B	A != B
Mayor que	A > B	
Menor que	A < B	
Mayor o igual que	A >= B	
Menor o igual que	A <= B	A <= B
Operador lógico AND	A = B and C = D	A = B & C = D
Operador lógico ORº	A = B or C = D	A = B | C = D
Operador lógico OR de cortocircuito	NOT A = B	A ^ B
Operador lógico AND de cortocircuito	A = B AND NOT C = D	A = B && C = D
Operador lógico NOT	A NOT B	A ! B

***Tabla 1.** Similitudes y diferencias en los operadores en VB.NET y C#.*



ALCANCES DEL LENGUAJE

Si bien C# es parte del framework y plataforma .NET, debemos reconocerlo también como un lenguaje de programación independiente, ya que existe un compilador basado en el framework DotGNU, conocido como Mono, que permite generar programas tanto para Win32, como para UNIX y Linux. Más información en **www.mono-project.com**.

La declaración de variables siempre es obligatoria antes de su uso, al igual que en todo lenguaje orientado a objetos, y al momento de declararla, ya se le puede asignar un valor. Los tipos de datos en C# son iguales a los utilizados en Visual Basic. Veamos a continuación un ejemplo:

```
//Declaración de una variable string

String strTituloVentana;

//Y asignándole un valor en el momento de su declaración sería:

String strTituloVentana = "ABM Conductores";
```

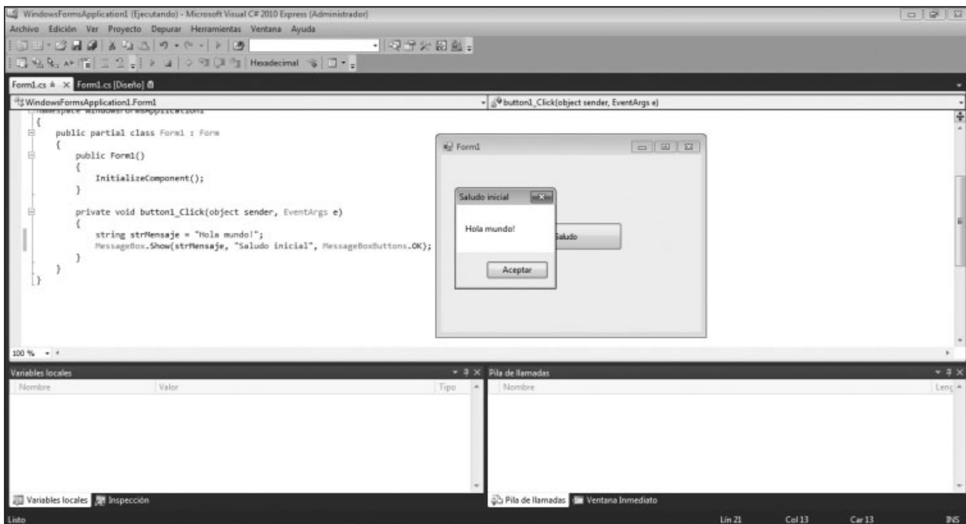


Figura 3. *El clásico "Hola mundo", aplicando la creación de una variable junto con la asignación de su valor, realizado en Visual C#.*



SIMILITUD EN INSTRUCCIONES

Las instrucciones **if-else**, **Switch**, **For**, **While**, **Do While**, **Foreach**, **break**, **Continue** y **Return** se utilizan de la misma manera en C# que otros lenguajes similares, como C, C++ y Java. **Goto** también se puede usar, aunque la polémica con respecto a ella es muy grande, dado que únicamente solía emplearse en lenguajes no orientados a objetos.

Todo código dentro de eventos, funciones y procedimientos en una aplicación C# lleva llaves (**{ }**) que indican su inicio y fin, como así también se aplican en el inicio y fin de una instrucción. Los comentarios que se deben realizar dentro de la aplicación o en aquellos casos en que hay que comentar un código para que no se ejecute, se efectúan con una doble contrabarra (**//**) al inicio.
En caso de que debamos comentar varias líneas de código, se inicia el comentario con una contrabarra seguida de un asterisco (**/***), y al final se cierra con un asterisco seguido de una contrabarra (***/**), tal como vemos en la **Figura 4**.

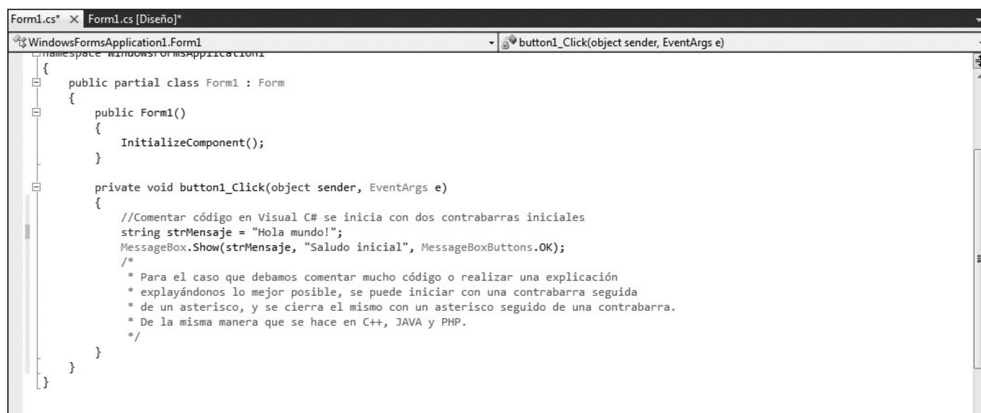


***Figura 4.** Los comentarios de código en C# también poseen similitudes con C, C++ y Java.*

Todos los **namespaces** utilizados en Visual Basic 2010 son visibles en Visual C#, con lo cual, habiendo aprendido sobre ellos, comenzar a programar en C# nos hará sentir más cómodos, ya que tendremos el principal conocimiento de namespaces y sabremos cómo aplicarlos en el lenguaje.

SINTAXIS DEL LENGUAJE C#

Veamos a continuación cómo se aplican las instrucciones más comunes dentro de C#. Para comprender claramente las mismas, haremos una comparación entre una instrucción en Visual Basic y su equivalente en este lenguaje. La instrucción **if** en Visual Basic se utiliza de la siguiente manera:

```
'Visual Basic

If coneccionRealizada Then
    MessageBox.Show("Se ha conectado a la base de datos.")
```

```
End If


// Visual C#

If (conexionRealizada == true)
     {
     MessageBox.Show("Se ha conectado a la base de datos.");
}
```

Veamos ahora cómo realizar una **operación aritmética**:

```
'Visual Basic

Public Sub operación()
Dim n as integer = 35
Dim o as integer = 38
Dim result as integer = n * o
End sub

// Visual C#

Public void Main()
{
int n = 35;
int o = 38;
int result = n * o;
}
```

Para agregar un namespace a nuestro proyecto:

```
' Visual Basic

Imports System.IO

// Visual C#
Using System.IO;
```

```
. . .
 private void button1_Click(object sender, EventArgs e)
 {
     StreamReader srder;
     srder = new StreamReader();
     string strLinea;
     string rutaArchivo = “C:\\cluf.txt”;
     strLinea = srder.ReadLine();


... // más código

 }
```

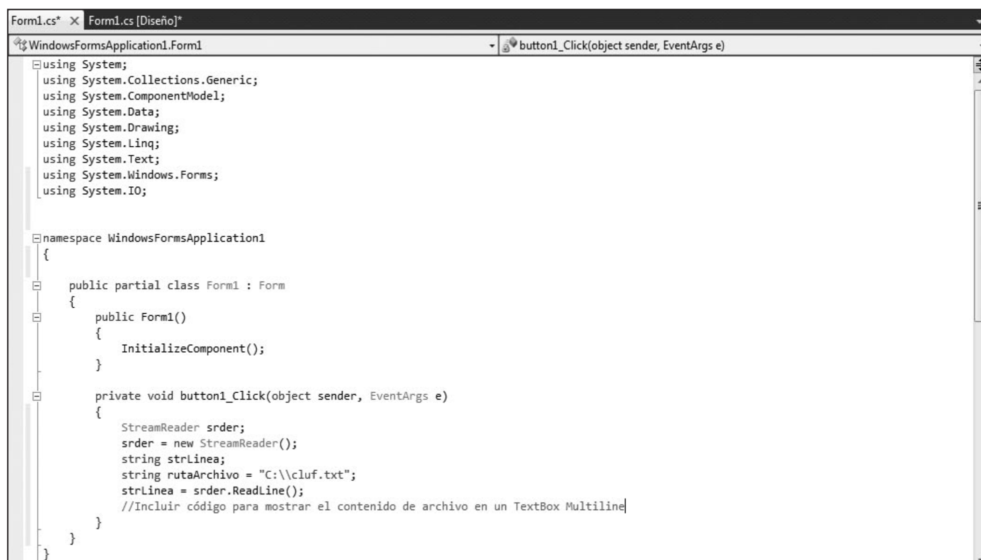


Figura 5. *Especificar un **Path** en C# requiere incluir una doble barra en la ruta, dado que los **strings** utilizan **caracteres de escape**.*



MATRICES EN C#

C# también permite el uso de matrices. La sintaxis de uso correspondiente es **tipo[] nombre_de_matriz = new tipo[nuevotamaño];**. Y si deseamos utilizar una matriz bidimensional, debemos optar por el siguiente formato: **tipo[,] nombre_de_matriz = new tipo[fila_matriz, columna_matriz];** a diferencia de C++, que usa **nombre_de_matriz[fila] [columna]**.

Para poder realizar conexiones a bases de datos, disponemos del mismo asistente de conexión que en Visual Basic, con lo cual no nos será complicado tener que mostrar en una grilla los datos almacenados en una tabla de Access, SQL Server o cualquier otro motor de base de datos.

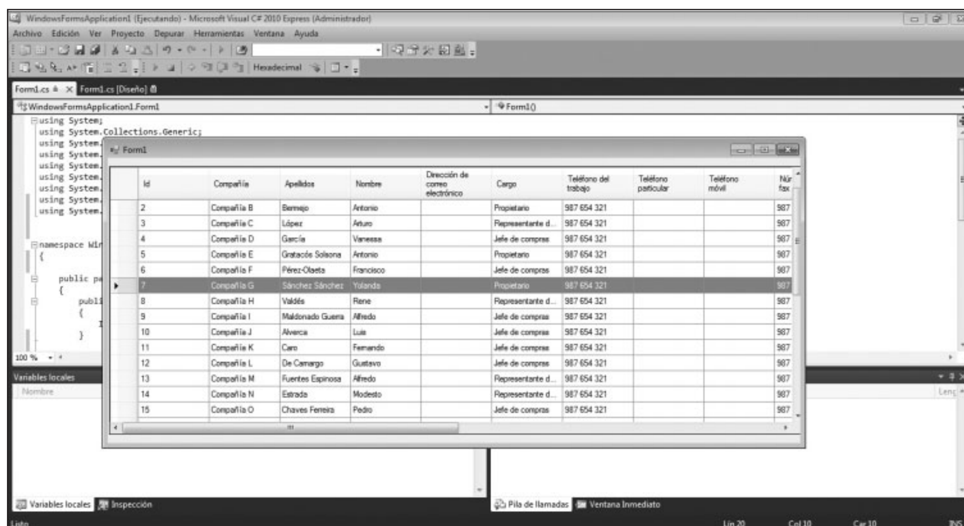


Figura 6. *Utilizando el Asistente de conexión, se estableció el enlace con la base de datos* ***Northwind 2007*** *y se está visualizando la tabla* **Clientes** *en un* **DataGridView**.

También podremos utilizar la declaración de los objetos correspondientes a bases de datos y tablas, tal como lo hacemos en Visual Basic. Las instrucciones son las mismas, solo cambia la manera de declarar las variables y el punto y coma que debemos incluir al final de cada instrucción escrita.

```
// Debemos incluir en principio las referencias a los namespaces

using System;
using System.Data;
using System.Data.SqlClient;

// Luego se declara el ConecctionString dentro de una clase

class Class1 { static void Main(string[] args)
{
string sConnectionString = “User ID=<UID>;Initial Catalog=pubs;Data
Source=(local)”; }
}
```

Luego se llama al método **Open()** del objeto **connection**, el cual nos permitirá fácilmente establecer la conexión a la base de datos:

```
SqlConnection objConn = new SqlConnection(sConnectionString);
objConn.Open();
```

A continuación, hay que el objeto **sqlCommand** u **OleDBCommand**, dependiendo del tipo de conexión elegido. Veamos una instrucción **INSERT** de SQL:

```
string sSQL = "INSERT INTO Empleados " + "( id, name, lname, jid,
hire_date) " + "VALUES ('1234', 'Fernando', 'Luna', 'W', '1996-07-14')";
SqlCommand objCmd = new SqlCommand(sSQL,objConn);
```

Y, por último, se llama al comando **ExecuteNonQuery()** para procesar el **INSERT** creado en el paso anterior sin esperar un resultado a mostrar:

```
objCmd.ExecuteNonQuery();
```

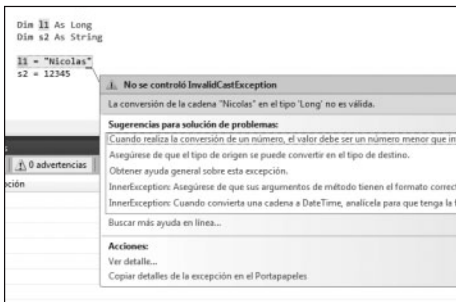
Como vimos hasta aquí, conectarse a una base de datos, cualquiera sea su tipo, y realizar operaciones sobre ella, no es para nada complicado en C#; de hecho, es muy similar a lo realizado en Visual Basic. Veamos en la **Tabla 2** algunos links recomendados para conocer un poco más el lenguaje C#.
Por último, podemos tener en cuenta también el libro **C#, La guía total del programador**, de esta misma editorial, en donde encontraremos cientos de ejercicios prácticos que nos ayudarán a comprender mejor el lenguaje.
Es altamente recomendable aprender más de un lenguaje de programación, lo que nos favorecerá al momento de buscar empleo o para ofrecernos como programadores. Dada la familiaridad de C# con Visual Basic, y ya que ambos comparten algo en común, que es el framework .NET, es más que lógico que, al aprender un nuevo lenguaje, sabiendo alguno de los dos primeramente, los tiempos de aprendizaje se reducirán de manera considerable. Es posible obtener más información del libro recomendado en el siguiente link: **www.redusers.com/noticias/libro-c-sharp-guia-total-del-programador**.

Servicios al lector

En esta última sección de este libro, encontraremos un índice que nos permitirá ubicar de forma más sencilla los temas y conceptos que necesitemos.

ÍNDICE TEMÁTICO

A



B

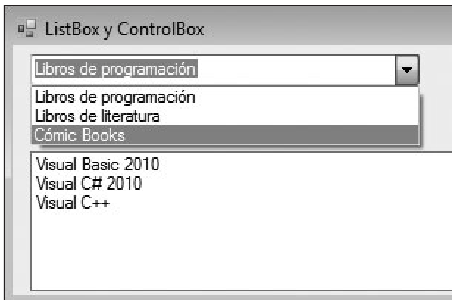
C

D

E

La configuración de la aplicación le permite almacenar y recuperar la c
información para su aplicación dinámicamente. Por ejemplo, la aplicac
del usuario y, a continuación, recuperarlas la próxima vez que se ejecut
configuración de la aplicación...

	Nombre	Tipo	Ámbito	Valor
▸	direccionip	String	Usuario	200.24
	usuario	String	Usuario	root
	clave	String	Usuario	miclav
*				

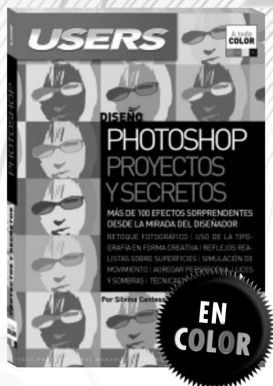

ListBox y ControlBox
Libros de programación
Libros de programación
Libros de literatura
Cómic Books
Visual Basic 2010
Visual C# 2010
Visual C++



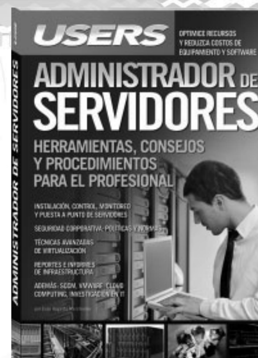


usershop.redusers.com









Photoshop: proyectos y secretos

En esta obra aprenderemos a utilizar Photoshop, desde la original mirada de la autora. Con el foco puesto en la comunicación visual, a lo largo del libro adquiriremos conocimientos teóricos, al mismo tiempo que avanzaremos sobre la práctica, con todos los efectos y herramientas que ofrece el programa.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / 978-987-1773-25-1

WordpPress

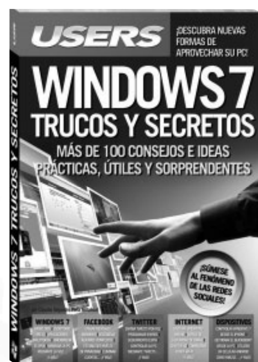
Este manual está dirigido a todos aquellos que quieran presentar sus contenidos o los de sus clientes a través de WordPress. En sus páginas el autor nos enseñará desde cómo llevar adelante la administración del blog hasta las posibilidades de interacción con las redes sociales.

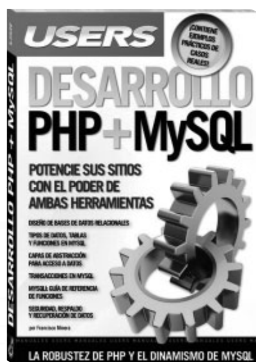
→ COLECCIÓN: MANUALES USERS
→ 352 páginas / 978-987-1773-18-3

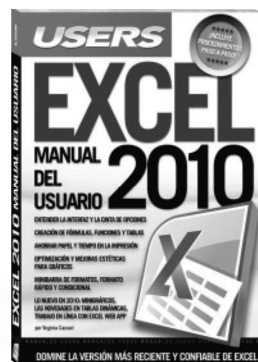
Administrador de servidores

Este libro es la puerta de acceso para ingresar en el apasionante mundo de los servidores. Aprenderemos desde los primeros pasos sobre la instalación, configuración, seguridad y virtualización; todo para cumplir el objetivo final de tener el control de los servidores en la palma de nuestras manos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-19-0







Windows 7: Trucos y secretos

Este libro está dirigido a todos aquellos que quieran sacar el máximo provecho de Windows 7, las redes sociales y los dispositivos ultraportátiles del momento. A lo largo de sus páginas, el lector podrá adentrarse en estas tecnologías mediante trucos inéditos y consejos asombrosos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-17-6

Desarrollo PHP + MySQL

Este libro presenta la fusión de dos de las he rramientas más populares para el desarrollo de aplicaciones web de la actualidad: PHP y MySQL. En sus páginas, el autor nos enseñará las funciones del lenguaje, de modo de tener un acercamiento progresivo, y aplicar lo aprendido en nuestros propios desarrollos.

→ COLECCIÓN: MANUALES USERS
→ 432 páginas / ISBN 978-987-1773-16-9

Excel 2010

Este manual resulta ideal para quienes se inician en el uso de Excel, así como también para los usuarios que quieran conocer las nuevas herramientas que ofrece la versión 2010. La autora nos enseñará desde cómo ingresar y proteger datos hasta la forma de imprimir ahorrando papel y tiempo.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-15-2

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



Técnico Hardware

Esta obra es fundamental para ganar autonomía al momento de reparar la PC. Aprenderemos a diagnosticar y solucionar las fallas, así como a prevenirlas a través del mantenimiento adecuado, todo explicado en un lenguaje práctico y sencillo.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-14-5



PHP Avanzado

Este libro brinda todas las herramientas necesarias para acercar al trabajo diario del desarrollador los avances más importantes incorporados en PHP 6. En sus páginas, repasaremos todas las técnicas actuales para potenciar el desarrollo de sitios web.

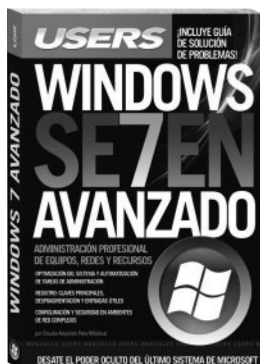
→ COLECCIÓN: MANUALES USERS
→ 400 páginas / ISBN 978-987-1773-07-7



AutoCAD

Este manual nos presenta un recorrido exhaustivo por el programa más difundido en dibujo asistido por computadora a nivel mundial, en su versión 2010. En sus páginas, aprenderemos desde cómo trabajar con dibujos predeterminados hasta la realización de objetos 3D.

→ COLECCIÓN: MANUALES USERS
→ 384 páginas / ISBN 978-987-1773-06-0



Windows 7 Avanzado

Esta obra nos presenta un recorrido exhaustivo que nos permitirá acceder a un nuevo nivel de complejidad en el uso de Windows 7. Todas las herramientas son desarrolladas con el objetivo de acompañar al lector en el camino para ser un usuario experto.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-08-4



Photoshop

En este libro aprenderemos sobre las más novedosas técnicas de edición de imágenes en Photoshop. El autor nos presenta de manera clara y práctica todos los conceptos necesarios, desde la captura digital hasta las más avanzadas técnicas de retoque.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-05-3



Grabación y producción de música

En este libro repasaremos todos los aspectos del complejo mundo de la producción musical. Desde las cuestiones para tener en cuenta al momento de la composición, hasta la mezcla y el masterizado, así como la distribución final del producto.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-04-6

usershop@redusers.com

USERS

CONTENIDO

NIVEL DE USUARIO

PRINCIPIANTE | INTERMEDIO | AVANZADO | EXPERTO

VISUAL BASIC

Este libro está escrito para aquellos usuarios que quieran aprender a programar en VB.NET, así como también para quienes provengan de otros lenguajes o necesiten actualizarse desde alguna versión antigua de Visual Basic. Aquí aprenderán a desenvolverse en el IDE de programación, desarrollar verdaderas aplicaciones RAD, conectarse y operar con bases de datos SQL Server, diseñar aplicaciones web y, por último, dar un vistazo al desarrollo para el nuevo Windows Phone 7.
Todos los procedimientos son expuestos de manera práctica con el código fuente de ejemplo (disponible en Internet), diagramas conceptuales y la teoría necesaria para comprender en detalle cada tema presentado.
Al finalizar el último capítulo, el lector conocerá en profundidad cómo programar en VB.NET, al mismo tiempo que convertirá el libro en una guía de consulta futura para concretar soluciones de software en pocos pasos.
Fernando Omar Luna es Analista Programador Universitario, experto en sistemas de gestión, y con su vasta experiencia, es el guía ideal para introducir al lector en el interesante mundo de la programación en VB.NET.





VISUAL BASIC

This manual aims to teach the fundamental aspects of Visual Basic programming. The reader will also learn about VB.NET's IDE, how to connect with SQL, and develop web apps, among many other tools and techniques to become a professional developer.



ISBN 978-987-1773-57-2